The Waite Group's

MS-DOS QBasic™

GUIDA DEL PROGRAMMATORE

Robert Arnson, Christy Gemmell, Harry Henderson

Microsoft® PRESS

McGraw Hill

MS-DOS QBasic™

GUIDA DEL PROGRAMMATORE

The Waite Group's

MS-DOS QBasic™
GUIDA DEL PROGRAMMATORE

Robert Arnson, Christy Gemmell, Harry Henderson

Microsoft PRESS®

McGRAW-HILL Libri Italia srl

Milano • New York • St. Louis • San Francisco • Oklahoma City
Auckland • Bogotà • Caracas • Hamburg • Lisboa
London • Madrid • Montreal • New Delhi • Paris • San Juan
São Paulo • Singapore • Sidney • Tokyo • Toronto

Ogni cura è stata posta nella creazione, realizzazione, verifica e documentazione dei programmi contenuti in questo libro. Tuttavia né l'Autore né la McGraw-Hill Libri Italia possono assumersi alcuna responsabilità derivante dall'implementazione dei programmi stessi, né possono fornire alcuna garanzia sulle prestazioni o sui risultati ottenibili dal loro uso, né possono essere ritenuti responsabili di danni o benefici risultanti dall'utilizzo dei programmi. Lo stesso dicasi per ogni persona o società coinvolta nella creazione, nella produzione e nella distribuzione di questo libro.

Titolo originale: *MS-DOS QBasic Programmer's reference*
Copyright © 1991 the Waite Group's

All right published by arrangement with the original publisher Microsoft Press, a division of Microsoft Corporation, Redmond, Washington, U.S.A.

Copyright © 1992 McGraw-Hill Libri Italia srl
piazza Emilia, 5
20129 Milano

I diritti di traduzione, di riproduzione, di memorizzazione elettronica e di adattamento totale e parziale con qualsiasi mezzo (compresi i microfilm e le copie fotostatiche) sono riservati per tutti i paesi.

Capo redattore: Massimo Esposti
Redattore: Chiara Tartara

Hanno collaborato a questo volume:
Realizzazione: Edimatica snc, via G.B. Morgagni 10, 20129 Milano
Traduzione: Laura Gaggini
Revisione tecnica: Sergio Zazzera
Copertina: Studio Grafico Margherita
Stampa: Arti Grafiche Battaia snc, Rozzano (MI)

ISBN 88-386-0257-3

Printed in Italy
1234567890GBTLLC965432
1ª edizione aprile 1992

Hayes è un marchio registrato della Hayes Microcomputer Products Inc.; Hewlett-Packard e LaserJet sono marchi registrati della Hewlett-Packard company; IBM, AT, Personal System/2, PS/2, PC/XT e AT sono marchi registrati della International Business Machines Corporation; Microsoft, MS-DOS, QBasic e Microsoft QuickBasic sono marchi registrati della Microsoft Corporation; UNIX è un marchio registrato della American Telephone and Telegraph Company.

Indice

Introduzione ... IX

PARTE PRIMA ELEMENTI FONDAMENTALI ... 1

Capitolo 1 Variabili e tipi ... 3
Introduzione ... 3
Nomi mnemonici per i valori delle costanti ... 4
Tipi di variabili ... 5
Tipi predefiniti ... 7
Conversione di un valore decimale in esadecimale od ottale ... 7
Assegnazione dei valori alle variabili ... 8
Scambio dei valori tra due variabili ... 8
Definizione dei tipi di variabile composti ... 8

Capitolo 2 Controllo del flusso ... 33
Introduzione ... 33
Salto ... 33
Cicli ... 35

Capitolo 3 Decisioni e operatori ... 55
Introduzione ... 55
IF - l'istruzione decisionale di carattere generale ... 55
Operatori relazionali ... 56
Gli operatori logici ... 59
SELECT CASE ... 60
L'operatore MOD ... 61
L'operatore + (concatenazione) ... 61

Capitolo 4 Procedure ... 83
Introduzione ... 83
Sottoprogrammi ... 84
Subroutine ... 86
Funzioni definite dall'utente ... 86
Concatenamento ... 87

Capitolo 5 Stringhe 115
Introduzione 115
Definizione e utilizzo delle stringhe 116
Funzioni di gestione delle stringhe 121
Conclusioni 125

Capitolo 6 Array e dati 149
Introduzione 149
Dichiarazione e utilizzo degli array 150
Array statici e dinamici 156

Capitolo 7 Funzioni matematiche 177
Introduzione 177
Funzioni trigonometriche e radice quadrata 177
Logaritmi ed esponenziali 180
Funzioni di conversione numerica 180
Troncamento e arrotondamento 181
Numeri casuali 181

Capitolo 8 Funzioni di I/O 205
Introduzione 205
Input da tastiera 205
Output del testo 207
Il cursore in modo testo 207
Dimensioni dello schermo 208
La finestra di testo 209
Cancellazione dello schermo 209

Capitolo 9 Gestione degli errori 241
Introduzione 241
Intercettazione degli errori 242
Gestori degli errori 243
Gestione degli eventi 244

Capitolo 10 Data e ora 259
Introduzione 259
Impostazione e visualizzazione della data 259
Impostazione e visualizzazione dell'ora 260
Sospensione del programma 261
Impostazione e gestione degli eventi del timer 261

PARTE SECONDA MULTIMEDIA 273

Capitolo 11 Grafica 275
Introduzione 275
Adattatori video 275
Pixel e risoluzione 277

Modalità video 277
Coordinate dello schermo 278
Disegni 278
Colori 280
Viewport grafico e finestre 280
Animazione 282
Pagine video 282
Visualizzazione dei grafici con l'istruzione draw 282

Capitolo 12 Suono 333
Introduzione 333
Utilizzo della musica 333
Traduzione di uno spartito in istruzioni play 334
La musica in sottofondo 339
Altre istruzioni relative al suono 340

Capitolo 13 La penna ottica e il joystick 351
Introduzione 351
Penne ottiche 351
Joystick 352
Impostazione della gestione degli eventi 352
I valori di posizione 354
Utilizzo del mouse come penna ottica 355

PARTE TERZA DISPOSITIVI 369

Capitolo 14 La tastiera 371
Introduzione 371
Tasti rapidi 371
Gestione degli eventi della tastiera 372

Capitolo 15 Stampanti 381
Introduzione 381
Stampa di una parte di testo 381
Controllo della stampante 383

Capitolo 16 Porte di comunicazione 391
Introduzione 391
Impostazione del collegamento 392
Caricamento e invio dei caratteri 394
Gestione delle comunicazioni con la gestione degli eventi 394
Altre informazioni 395

Capitolo 17 File 407
Introduzione 407
Apertura di un file o di un dispositivo 408
Chiusura di un file o di un dispositivo 411

Gestione dei file ad accesso casuale 412
Lettura dei dati dai file ad accesso casuale o binari 413
Scrittura dei dati nei file ad accesso casuale o binari 414
Lettura dei dati dai file sequenziali 414
Scrittura dei dati nei file sequenziali 415
Eliminazione degli spazi dalle stringhe 415
Spostamento all'interno di un file 415
Determinazione delle caratteristiche dei file 416
Gestione dei file su disco 416
Blocco dei file 417
Impostazione della larghezza massima dei file sequenziali 417

PARTE QUARTA SVILUPPO 483

Capitolo 18 Il DOS e la gestione dei programmi 485
Introduzione 485
Il separatore di istruzioni e i commenti 485
Uscita da un programma QBasic 487
Utilizzo dei comandi dos in un programma 487
Reinizalizzazione delle variabili e impostazione delle dimensioni dello stack 487
Lettura e impostazione delle variabili di ambiente 488
Comunicazione con i driver dei dispositivi 488
Gestione dei file e delle directory 489

Capitolo 19 Porte e memoria 513
Introduzione 513
Memoria 513
Porte 516

Capitolo 20 QBasic e il linguaggio macchina 535
Introduzione 535
Caricamento di routine in linguaggio macchina 535

Capitolo 21 Debug 547
Una sessione di debug 547

Appendice A Messaggi di errore 563
I messaggi di errore di QBasic 563
Errori di richiamo, di compilazione e di esecuzione 564

Indice analitico 583

Introduzione

Questo libro è stato scritto in base a un metodo ben preciso, che consente di ottenere innanzitutto le informazioni più generali, per poi passare a quelle più specifiche. Ad esempio, chi utilizza QBasic per la prima volta, oppure conosce già un'altra versione di Basic e desidera comprendere quali parti sono diverse, potrebbe voler leggere una panoramica generale di tutto il linguaggio, delle caratteristiche e dei limiti, così come delle possibilità. Tutte queste informazioni vengono presentate nelle parti iniziali dei capitoli, in modo comprensibile e chiaro.

Se dopo aver compreso il quadro generale, si deve invece utilizzare un comando, è necessario approfondire i diversi argomenti. Ogni capitolo esamina lo scopo, la sintassi e l'utilizzo di ogni istruzione e funzione, fornendo le informazioni relative a ogni voce, compresi alcuni suggerimenti per la programmazione e alcune avvertenze. L'analisi delle istruzioni e delle funzioni è stata strutturata per soggetto e non alfabeticamente; in questo modo è possibile comprendere le relazioni tra le istruzioni in un'area specifica e quindi afferrare velocemente le relative interazioni, senza dover cercare voci sparse in tutto il libro.

La struttura di questo libro è stata sviluppata organizzando tutte le parole chiave di QBasic in gruppi correlati. Quindi, ogni istruzione e funzione è stata esaminata e sono stati scelti gli esempi migliori per illustrarla. Poi, sono state scritte le pagine di riferimento per ogni comando. I brevi programmi di esempio sono stati progettati con lo scopo di illustrare i comandi in modo interessante, utile e chiaro. Infine, gli elementi più importanti di ogni categoria dei comandi sono stati riepilogati, in modo che fosse semplice comprendere come utilizzarli insieme.

In ogni fase del processo, le informazioni sono state scelte con discernimento, estraendo i fatti più importanti.

Si consideri questo libro come un microscopio, che dispone di diverse possibilità di ingrandimento. Si può iniziare al livello inferiore e quindi leggere le parti introduttive per apprendere le relazioni esistenti tra i comandi, passando infine a esaminare dettagliatamente ogni argomento nelle voci di riferimento.

Per coerenza con la sintassi del linguaggio Qbasic, nel libro i numeri vengono espressi nella notazione americana, per cui il numero "unmilione" equivale a 1,000,000 e "zero virgola cinque" equivale a 0.5.

Parte prima

Elementi fondamentali

Capitolo 1

Variabili e tipi

INTRODUZIONE

I programmatori che utilizzano QBasic hanno l'indubbia fortuna di potersi concentrare sulla soluzione dei problemi e la progettazione dei propri programmi, lasciando all'ambiente di sviluppo il compito di gestire i dettagli della programmazione.

QBasic consente di utilizzare nomi simbolici per le locazioni in corrispondenza delle quali vengono memorizzati i valori, denominate *variabili*, descrivendone il significato. Ad esempio, è possibile scegliere il nome di variabile ImportoScaduto per memorizzare l'importo di cui è debitore un cliente.

È importante scegliere nomi di variabili che siano autoesplicativi, in modo che sia il programmatore sia coloro che leggono i programmi siano in grado di riconoscere immediatamente il significato dei valori memorizzati. Alcuni abbreviano tali nomi per risparmiare il tempo necessario a digitarli, ma in questo modo si rischia di diminuire la chiarezza. Ad esempio, abbreviando il nome della variabile ImportoScaduto in ImpScd, lo scopo rimane chiaro; tuttavia, se il nome viene ulteriormente abbreviato in ImpS, successivamente si potrebbe essere portati a ritenere che significhi "importo del saldo". In QBasic i nomi delle variabili devono seguire le regole di seguito riportate.

- Il primo carattere deve essere una lettera (da A a Z), in maiuscolo o minuscolo.
- I caratteri restanti possono essere lettere (da A a Z) in maiuscolo o minuscolo, numeri da 0 a 9 oppure punti (.).
- L'ultimo carattere può essere un carattere speciale (%, &, !, # oppure $) che indichi il tipo di valore da memorizzare nella variabile.
- La lunghezza massima è fissata in 40 caratteri.
- Non è consentito utilizzare i nomi delle parole riservate di QBasic.

Il nome di una variabile può essere considerato come la targhetta presente sulla porta di un ufficio, che indica la funzione svolta dalla persona che lo occupa, ma non il suo nome; in effetti la persona che esegue quella funzione può cambiare senza che

per questo debba essere modificata la targhetta. Ad esempio, la variabile presidente$ potrebbe comprendere il valore "Rossi", ma nel futuro il valore di questa variabile potrebbe essere "Bianchi"; il nome non viene modificato, ma cambia il contenuto.

Quando vengono create le variabili di un programma, è necessario scegliere tra due tipi distinti: *numeriche* e *carattere.* In QBasic, come nella maggior parte dei linguaggi di programmazione, una variabile numerica può essere di tipi diversi, ognuno dei quali definisce un intervallo di valori che la variabile può contenere; può essere un *intero,* un *intero lungo,* un valore *a precisione semplice* oppure *a precisione doppia.* Una variabile carattere può essere di due tipi: una *stringa a lunghezza variabile* o *a lunghezza fissa.* Le istruzioni e le funzioni descritte in questo capitolo possono essere utilizzate per definire le variabili e convertire i valori in tipi diversi. Nel Capitolo 5 viene affrontato l'argomento delle stringhe.

In questa parte vengono riepilogate le variabili e i tipi e vengono offerti alcuni suggerimenti relativi al loro utilizzo. Nella Tabella 1.1 sono riportate in ordine alfabetico le parole chiave, le istruzioni e le funzioni utilizzate per definire e gestire le variabili.

Tabella 1.1 Parole chiave delle variabili, istruzioni e funzioni

Parola chiave, istruzione o funzione	Descrizione
CONST	Definisce un nome mnemonico per i valori che non cambiano
DEFtipo	Definisce i caratteri che indicano i tipi di variabile di default
DOUBLE	Crea una variabile a precisione doppia
HEX$	Restituisce una stringa contenente il valore esadecimale corrispondente a un valore intero
INTEGER	Crea una variabile di tipo intero
LET	Assegna un valore a una variabile
LONG	Crea una variabile di tipo intero lungo
OCT$	Restituisce una stringa contenente il valore ottale corrispondente a un valore intero
SINGLE	Crea un valore a precisione singola
STRING	Crea una variabile stringa
SWAP	Scambia i valori di due variabili
TYPE...END TYPE	Definisce un nuovo tipo di variabile

NOMI MNEMONICI PER I VALORI DELLE COSTANTI

Dato che spesso la programmazione comporta l'utilizzo di codici numerici, QBasic dispone dell'istruzione CONST che consente di assegnare a essi nomi significativi, definiti *costanti.* L'istruzione COLOR, ad esempio, accetta come parametri valori compresi nell'intervallo da 0 a 31; ogni numero indica un colore diverso. I valori da 16 a 31 indicano gli stessi colori dei valori da 0 a 15, con l'aggiunta dell'attributo

lampeggiante. Possono essere definite delle costanti per rappresentare alcuni di questi valori, come riportato nelle seguenti istruzioni:

```
CONST NERO = 0, BLU = 1, VERDE = 2, CIANO = 3
CONST ROSSO = 4, MAGENTA = 5, MARRONE = 6, BIANCO = 7
CONST LAMPEGGIANTE = 16
```

Queste istruzioni consentono di modificare velocemente i colori sullo schermo senza che sia necessario memorizzare tutta la tabella di codici, ad esempio:

```
COLOR BIANCO, BLU
PRINT "Questo testo è bianco su sfondo blu"

COLOR LAMPEGGIANTE + ROSSO, BIANCO
PRINT "Questo testo è rosso lampeggiante su sfondo bianco"
```

L'istruzione **CONST** è inoltre utile quando si tratta di assegnare dei nomi a costanti che potrebbero essere successivamente modificate. Sebbene lo schermo standard visualizzi per default 25 righe di testo, le schede grafiche EGA supportano anche 43 righe di testo e quelle VGA consentono di visualizzarne 25, 43 o 50. Per consentire agli utenti che dispongono di queste schede di trarre il massimo vantaggio dalla presenza di un numero superiore di righe sullo schermo, è possibile definire una costante ALTEZZA, che dovrebbe essere impostata a 25 per lo schermo standard, a 43 per lo schermo EGA e a 50 per lo schermo VGA.

In QBasic il valore di una costante può essere sia un valore che un'espressione. Qualsiasi numero o stringa di caratteri può essere considerato come il valore di una costante; il numero 69 è una costante di tipo intero, mentre la stringa di caratteri "Guida di riferimento a QBasic" è una costante stringa. Un'espressione che contiene soltanto elementi costanti viene definita *espressione costante*; ad esempio:

```
69 + 182 * 2
```

è un'espressione costante, dato che ogni elemento è un valore costante. Tuttavia, l'espressione:

```
69 + 128 * FATTORE
```

è un'espressione costante soltanto se FATTORE è stato definito come valore costante; se FATTORE è una variabile, l'espressione riportata in precedenza non è una costante.

Le costanti definite con un'espressione possono contenere soltanto numeri, stringhe letterali e altre costanti, mentre non possono comprendere variabili o funzioni. Ad esempio, è possibile definire la costante ROSSOLAMPEGGIANTE nel modo riportato di seguito, dato che è una combinazione di altre costanti.

```
CONST ROSSO = 4, LAMPEGGIANTE = 16
CONST ROSSOLAMPEGGIANTE = ROSSO + LAMPEGGIANTE
```

TIPI DI VARIABILI

QBasic fornisce cinque tipi di variabili semplici: intero, intero lungo, a precisione semplice, a precisione doppia e stringa. Una variabile può essere dichiarata come uno

di questi tipi in una clausola AS nelle istruzioni QBasic riportate nella Tabella 1.2. Per ulteriori informazioni riguardanti queste istruzioni, si consiglia di consultare le relative voci. Il tipo di variabile può essere dichiarato in due modi, ovvero con una clausola AS oppure con un *carattere di dichiarazione del tipo.* Per utilizzare il primo metodo è sufficiente inserire dopo il nome di variabile la parola chiave AS e la parola chiave corrispondente al tipo desiderato. Ad esempio, l'istruzione di seguito riportata dichiara la variabile totale come tipo a precisione doppia.

```
COMMON totale AS DOUBLE
```

Nella Tabella 1.3 sono riportate le clausole AS per i cinque tipi semplici.

L'altro metodo è quello di inserire dopo il nome di variabile uno dei caratteri di dichiarazione del tipo: %, &, !, #, o $. A questo punto QBasic assegna un tipo a quella variabile, in base a quanto riportato nella Tabella 1.4.

Tabella 1.2 Le istruzioni di QBasic in cui è possibile dichiarare le variabili

COMMON	DIM	STATIC
DECLARE SUB	FUNCTION	SUB
DECLARE FUNCTION	REDIM	TYPE...END TYPE
DEF FN	SHARED	

Tabella 1.3 Dichiarazioni dei tipi con la parola chiave AS

Dichiarazione del tipo	Scopo
AS INTEGER	Dichiara una variabile di tipo intero
AS LONG	Dichiara una variabile di tipo intero lungo
AS SINGLE	Dichiara una variabile a precisione semplice
AS DOUBLE	Dichiara una variabile a precisione doppia
AS STRING	Dichiara una variabile stringa

Tabella 1.4 Caratteri di dichiarazione dei tipi

Carattere di dichiarazione del tipo	Scopo
%	Dichiara una variabile di tipo intero
&	Dichiara una variabile di tipo intero lungo
!	Dichiara una variabile a precisione semplice
#	Dichiara una variabile a precisione doppia
$	Dichiara una variabile stringa

Ad esempio, l'istruzione:

```
totale# = 128.69
```

assegna il valore 128.69 alla variabile a precisione doppia denominata totale#. È necessario tenere presente che il carattere di dichiarazione del tipo fa parte del nome di variabile e quindi è necessario inserirlo sempre in riferimento al nome. Si prenda il considerazione il seguente esempio:

```
totale# = 128.69
totale = 100.55
```

Si osservi che la variabile totale, senza il carattere di dichiarazione del tipo, è diversa dalla variabile totale#. In effetti, è possibile sfruttare un solo nome per cinque diverse variabili:

```
totale%      ' una variabile di tipo intero
totale&      ' una variabile di tipo intero lungo
totale!      ' una variabile a precisione semplice
totale#      ' una variabile a precisione doppia
totale$      ' una variabile stringa
```

Il carattere di dichiarazione del tipo consente di riconoscere il tipo di una variabile; ad esempio è facile ricordare che una variabile denominata totale# contiene un valore a precisione doppia.

TIPI PREDEFINITI

QBasic non richiede di dichiarare esplicitamente una variabile; invece di utilizzare AS o un carattere di dichiarazione del tipo, è possibile avvalersi dell'istruzione DEFtipo, che imposta il tipo di dati predefinito. La prima volta che viene utilizzata una nuova variabile, QBasic le assegna il tipo specificato mediante tale istruzione.

Queste istruzioni indicano lettere o intervalli di lettere. A qualsiasi nome di variabile che inizi con le lettere specificate e non termini con un carattere di dichiarazione del tipo, viene assegnato il tipo specificato dalla parola chiave DEFtipo, che può essere DEFINT, DEFLNG, DEFSNG, DEFDBL o DEFSTR. Ad esempio, nella parte di programma di seguito riportata alla variabile totale viene assegnato il tipo intero.

```
DEFINT M-Z
totale=12
```

CONVERSIONE DI UN VALORE DECIMALE IN ESADECIMALE OD OTTALE

Quando il valore di una variabile viene visualizzato mediante l'istruzione PRINT, esso viene rappresentato sotto forma di numero decimale (base 10); QBasic dispone tuttavia di due funzioni che consentono di visualizzare i numeri in altre basi. La funzione HEX$ converte un numero decimale in una stringa che rappresenta il valore

in base 16 (esadecimale), mentre OCT$ lo converte in una stringa che rappresenta il valore in base 8 (ottale). La rappresentazione dei numeri in esadecimale e ottale può risultare utile quando si tratta di gestire routine scritte in linguaggio assembler; in ogni caso, queste funzioni non vengono utilizzate frequentemente.

ASSEGNAZIONE DEI VALORI ALLE VARIABILI

L'utilità delle variabili sarebbe minima, se non fosse possibile assegnare loro dei valori. Spesso, ad esempio, è necessario incrementare il valore di una variabile che agisce come contatore; per realizzare questa operazione, è sufficiente utilizzare la seguente istruzione:

```
contatore = contatore + 1
```

Alle variabili stringa è possibile assegnare costanti o altre variabili stringa, oltre al risultato di un'operazione di concatenazione:

```
linguaggio$ = "QBasic "
titolo$ = "Guida di riferimento" + linguaggio$
```

Dopo l'esecuzione di queste istruzioni, la variabile titolo$ contiene il valore "Guida di riferimento QBasic".

SCAMBIO DEI VALORI TRA DUE VARIABILI

L'istruzione SWAP consente di scambiare i valori tra due variabili diverse e risulta utile soprattutto in alcuni algoritmi di ordinamento; equivale a tre istruzioni di assegnazione organizzate in una sola istruzione. Ad esempio:

```
SWAP globTotale, sottoTotale
```

scambia il valore delle variabili globTotale e sottoTotale ed equivale alle seguenti istruzioni:

```
temp = sottoTotale
sottoTotale = globTotale
globTotale = temp
```

DEFINIZIONE DEI TIPI DI VARIABILE COMPOSTI

Generalmente alcune voci vengono raggruppate, ad esempio il mese, il giorno e l'anno formano la data; il titolo, il nome e il cognome formano il nome completo. Probabilmente, il metodo più semplice per memorizzare i tre elementi separati relativi alla data consiste nell'utilizzare tre variabili: mese, giorno e anno. QBasic presenta l'istruzione TYPE...END TYPE, che consente di creare variabili di tipo *record* costituite da diversi elementi.

Per creare una struttura record relativa alle parti che compongono il nome di una persona è necessario utilizzare l'istruzione TYPE..END TYPE, come nel listato di seguito riportato.

```
TYPE NomeCompleto
   titolo AS STRING * 4
   nome AS STRING * 16
   cognome AS STRING * 20
END TYPE
```

Dopo aver creato la struttura precedente, è possibile definire una nuova variabile con quel tipo di struttura:

```
DIM nomeCliente AS NomeCompleto
```

A questo punto è possibile assegnare i valori alle variabili nella variabile record nomeCliente, come nelle seguenti istruzioni:

```
nomeCliente.titolo = "Sig."
nomeCliente.nome = "Giorgio"
nomeCliente.cognome = "Pettinati"
```

È inoltre possibile creare array di variabili record. Il listato di seguito riportato dichiara un array di variabili record che vengono definite dal tipo NomeCompleto e quindi assegna loro dei valori.

```
TYPE NomeCompleto
   titolo AS STRING * 4
   nome AS STRING * 16
   cognome AS STRING * 20
END TYPE

DIM nomeCliente(2) AS NomeCompleto

nomeCliente(1).titolo = "Sig."
nomeCliente(1).nome = "Giorgio"
nomeCliente(1).cognome = "Pettinati"

nomeCliente(2).titolo = "Sig.ra"
nomeCliente(2).nome = "Claudia"
nomeCliente(2).cognome = "Rigoni"
```

Nella Figura 1.1 è riportata la rappresentazione grafica dell'array delle variabili record dichiarato in precedenza. È inoltre possibile definire strutture che contengono altre strutture come membri. Questa tecnica risulta utile per poter nuovamente sfruttare le strutture di base già definite, ad esempio la struttura precedente per i nomi; altre strutture, più complicate, potrebbero contenere queste strutture di base. Il listato di seguito riportato, ad esempio, consente di dichiarare un array di variabili record del tipo definito Degenti. La struttura Degenti contiene la variabile nomeDegenti, ovvero la variabile record del tipo NomeCompleto.

```
TYPE NomeCompleto
   titolo AS STRING * 4
   nome AS STRING * 16
   cognome AS STRING * 20
END TYPE

TYPE Degenti
   nomeDegenti AS NomeCompleto
```

nomeCliente(1).

titolo	Sig.
nome	Giorgio
cognome	Pettinati

nomeCliente(2).

titolo	Sig.ra
nome	Claudia
cognome	Rigoni

Figura 1.1 I valori in un array di variabili record

```
   indirizzo AS STRING * 128
END TYPE

DIM degente(100) AS Degenti

degente(1).nomeDegenti.titolo = "Sig.ra"
degente(1).nomeDegenti.nome = "Anna"
degente(1).nomeDegenti.cognome = "Rossi"
degente(1).indirizzo = "via Della Quercia 16"

degente(2).nomeDegenti.titolo = "Sig."
degente(2).nomeDegenti.nome = "Giorgio"
degente(2).nomeDegenti.cognome = "Orsi"
degente(2).indirizzo = "via Dritta 45"
```

Nella Figura 1.2 è riportata una rappresentazione grafica della struttura presente all'interno della struttura delle variabili record precedenti.

degente(1).

nomeDegenti.	titolo	Sig.ra
	nome	Anna
	cognome	Rossi
indirizzo	via Della Quercia 16	

degente(2).

nomeDegenti.	titolo	Sig.
	nome	Giorgio
	cognome	Orsi
indirizzo	via Dritta 45	

Figura 1.2 I valori in una struttura all'interno di una struttura

L'istruzione TYPE...END TYPE consente di specificare un'ampia gamma di tipi per gli elementi, fornendo in questo modo ai programmatori grande flessibilità.

Bibliografia

Lafore, Robert. *The Waite Group's Microsoft C Programming for the PC*, rev. ed. Indianapolis, Ind. Howard W. Sams and Co., 1989.

Norton, Peter, and Richard Wilton. *The New Peter Norton Programmer's Guide to the IBM PC and PS/2*. Redmond, Wash.: Microsoft Press, 1988.

CONST

Scopo

L'istruzione CONST consente di fornire nomi simbolici e significativi ai valori delle costanti numeriche e stringa. Ad esempio, è possibile utilizzare l'istruzione CONST per assegnare il nome PI al numero 3.141592653.

Sintassi

CONST nomecostante = espressione [, nomecostante = espressione...]

nomecostante è il nome della costante, che deve essere diverso dal nome di qualsiasi parola chiave, variabile o sottoprogramma.

espressione è un'espressione numerica o di stringa contenente soltanto un valore costante o un'espressione costante.

Utilizzo

```
CONST PI = 3.141592653
```

Assegna il nome PI al valore 3.141592653.

```
CONST FALSO = 0, VERO = NOT FALSO
```

Definisce due costanti simboliche, VERO e FALSO.

Descrizione

La programmazione in QBasic richiede spesso l'utilizzo di codici numerici, che possono rappresentare qualsiasi valore, dai colori dello schermo ai tasti della tastiera; l'istruzione CONST consente di assegnare a questi codici nomi significativi.

La costante è la parola a cui si desidera assegnare l'espressione specificata. È possibile utilizzare alla fine del nome un carattere di dichiarazione del tipo (%, &, !, # o $) per specificare il tipo di costante, ma questo carattere non viene considerato parte del nome. Ad esempio, specificando:

```
CONST DIMENSIONEARRAY% = 2500
```

per assegnare il valore 2500 a DIMENSIONEARRAY%, è possibile utilizzare la costante senza dover inserire il carattere di dichiarazione del tipo, come nelle istruzioni di seguito riportate.

```
DIM degenti(100 TO DIMENSIONEARRAY)
FOR i% = 1 TO DIMENSIONEARRAY
```

Se il carattere di dichiarazione del tipo viene omesso, QBasic assegna alla costante un tipo che corrisponde a quello dell'espressione specificata. Se ad esempio l'espressione è "linguaggioverboso", QBasic immette la costante sotto forma di stringa; se l'espressione è numerica, QBasic determina il tipo più piccolo che la può contenere e assegna un valore di tipo intero, intero lungo, a precisione semplice o a precisione doppia. Ad esempio, nell'istruzione:

```
CONST VENDITEANNUALI = 298500
```

QBasic determina innanzitutto che 298500 non rientra nell'intervallo del tipo intero e quindi controlla se il valore è compreso nell'intervallo del tipo intero lungo; una volta determinato che si trova in questo intervallo, VENDITEANNUALI viene immesso come costante di tipo intero lungo.

Nell'istruzione di seguito riportata, 0.015 è un numero a virgola mobile.

```
CONST COSTOSERVIZIO = 0.015
```

Dato che la costante non può essere di tipo intero o intero lungo, QBasic verifica soltanto gli intervalli dei tipi a precisione semplice o precisione doppia. Dato che rientra nell'intervallo del tipo a precisione semplice, COSTOSERVIZIO diventa una costante a precisione semplice.

L'espressione specificata può essere un'espressione QBasic che contiene una stringa, costanti numeriche, altre costanti definite mediante l'istruzione CONST e gli operatori matematici e logici, tranne l'elevamento a potenza; non può contenere variabili, funzioni definite dal programma oppure funzioni e istruzioni QBasic, quali CHR$, SIN e SEEK.

Prima di utilizzare una costante è necessario definirla con l'istruzione CONST. L'istruzione di seguito riportata, ad esempio, genera un errore di esecuzione, dato che si è utilizzata la costante FALSO prima di definirla.

```
CONST VERO = NOT FALSO, FALSO = 0
```

Il metodo corretto per definire le costanti simboliche VERO e FALSO è il seguente:

```
CONST FALSO = 0, VERO = NOT FALSO
```

Se una costante viene definita mediante l'istruzione CONST all'interno di un sottoprogramma o di una funzione definita dall'utente, è possibile utilizzarne il nome soltanto all'interno del sottoprogramma o della funzione stessa. Le costanti di questo tipo, dette definite *locali*, devono essere utilizzate quando si devono applicare soltanto a un sottoprogramma o a una funzione. Se ad esempio viene creato un sottoprogramma per tracciare il numero di transazioni completate, mediante l'istruzione di seguito riportata si crea una costante locale.

```
CONST MAXTRANSAZIONI =1024
```

Se invece una costante viene definita mediante l'istruzione CONST all'interno del programma principale è possibile utilizzarne il nome in tutto il programma. Le costanti di questo tipo, definite *globali*, quali VERO e FALSO, possono essere utilizzate quando devono essere applicabili in tutto un programma.

Una costante può essere utilizzata esattamente come una variabile o un'espressione.

Commenti

Il tipo di una costante non viene influenzato dalle istruzioni DEFtipo (DEFDBL, DEFINT, DEFLNG, DEFSNG e DEFSTR); può essere specificato soltanto aggiungendo un carattere di dichiarazione del tipo alla fine del suo nome, ad esempio:

```
DEFDBL A-Z

CONST IMPORTOMINIMO = 125.50
ImportoMassimo& = 9999
```

IMPORTOMINIMO è una costante a precisione semplice, mentre ImportoMassimo& è una variabile di tipo intero lungo.

Errori

Se viene creata un'espressione che non può essere una costante, QBasic visualizza il messaggio di errore: "Costante non valida". Se si cerca di modificare il nome di una costante definita in un'istruzione CONST, QBasic visualizza il messaggio di errore: "Duplice definizione". Se viene utilizzato l'operatore dell'elevamento a potenza (^) in un'espressione contenuta in un'istruzione CONST, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita".

Suggerimenti

L'utilizzo dell'istruzione CONST offre diversi vantaggi rispetto all'utilizzo delle variabili o dei letterali.

- L'utilizzo delle costanti consente di verificare i significati dei valori specifici, invece dei valori stessi. L'utilizzo della costante F1 per rappresentare il valore relativo al tasto funzione F1, ad esempio, risulta più chiaro rispetto all'utilizzo del valore 59.
- Quando una costante viene definita all'esterno di un sottoprogramma o di una funzione definita dall'utente, è disponibile per tutto il programma; a queste costanti è possibile accedere soltanto con l'istruzione SHARED.
- Il valore di una costante definita in un'istruzione CONST non può essere modificato accidentalmente, cosa che può invece accadere per il valore di una variabile.
- L'utilizzo delle costanti dichiarate in un'istruzione CONST consente di apportare modifiche al programma più facilmente. Se in origine un programma è stato progettato per gestire 25 righe sullo schermo, si potrebbe dichiarare la costante RIGHESCHERMO nel seguente modo:

  ```
  CONST RIGHESCHERMO = 25
  ```

  In seguito, per ampliare il programma in modo che sia in grado di gestire lo schermo VGA con 50 righe, è sufficiente modificare l'istruzione nel seguente modo:

  ```
  CONST RIGHESCHERMO = 50
  ```

È pratica comune digitare in lettere maiuscole i nomi delle costanti, per distinguerle dalle variabili.

Esempio

Il programma di seguito riportato utilizza l'istruzione CONST per fornire nomi simbolici relativi ai codici ASCII di determinate sequenze di tasti.

```
'definisce costanti che contengono i codici ASCII di alcuni tasti
CONST ESC = 27, INVIO = 13, CTRLBACKSPACE = 127

CLS
PRINT "Inserisci qualcosa ... (premi Ctrl + Backspace per terminare)."
PRINT
LOCATE , , 1                    'visualizza il cursore
DO
   DO                           'ripeti finchè è premuto un tasto

        inky$ = INKEY$
   LOOP WHILE inky$ = ""

   SELECT CASE ASC(inky$)
        CASE INVIO              'se il tasto è Invio, stampa [INVIO]
            PRINT " [INVIO]"

        CASE ESC                'se il tasto è Esc, stampa [ESC]
            PRINT " [ESC] ";

        CASE CTRLBACKSPACE      'se il tasto è Ctrl+Backspace, esce dal ciclo
            PRINT " [CTRL+BACKSPACE] ";
            EXIT DO

        CASE ELSE               'altrimenti, visualizza il tasto
            PRINT inky$;
   END SELECT
LOOP
```

DEFtipo

Si consulti anche

DIM, DOUBLE, INTEGER, LONG, SINGLE

Scopo

Le istruzioni DEFtipo di QBasic consentono di impostare il tipo di dati predefinito per le variabili e le funzioni definite dall'utente che non comprendono un carattere di dichiarazione del tipo o che non vengono specificatamente dichiarate da una clausola AS.

Sintassi

```
DEFDBFL intervallolettere [, intervallolettere...]
DEFINT intervallolettere [, intervallolettere...]
DEFLNG intervallolettere [, intervallolettere...]
DEFSNG intervallolettere [, intervallolettere...]
DEFSTR intervallolettere [, intervallolettere...]
```

intervallolettere è una lettera singola o un intervallo di lettere specificate da due lettere separate da un trattino.

Utilizzo

```
DEFLNG S, T
```

Dichiara tutte le variabili che iniziano con la lettera S o con la T di tipo intero lungo.

```
DEFINT A-Z
```

Dichiara tutte le variabili di tipo intero.

Descrizione

QBasic non richiede di definire il tipo e le dimensioni delle variabili e delle funzioni prima di utilizzarle. Se non viene specificato alcun tipo, QBasic ne assegna uno alla variabile la prima volta che viene utilizzata; questo consente di creare una nuova variabile ogniqualvolta sia necessario, senza doverla prima dichiarare. Le istruzioni DEFtipo impostano i tipi di dati predefiniti che vengono utilizzati quando QBasic assegna automaticamente un tipo alla variabile; quest'ultimo dipende dalla parola chiave DEFtipo utilizzata. Le cinque parole chiave sono riportate nella Tabella 1.5.

A qualsiasi variabile che non sia ancora stata definita e il cui nome inizia con una lettera specificata nell'istruzione DEFtipo, viene assegnato il tipo indicato dalla parola chiave DEFtipo (si consulti la Tabella 1.5.). Non si fa distinzione tra le variabili che iniziano con le lettere specificate in maiuscolo o minuscolo.

Tabella 1.5 Le parole chiave relative a DEFtipo

Con l'istruzione DEFtipo	Il tipo di variabile predefinita è
<nessuna>, DEFSNG	Precisione semplice
DEFDBL	Precisione doppia
DEFINT	Intero
DEFLNG	Intero lungo
DEFSTR	Stringa

Commenti

Dato che i tipi di tutti gli elementi in una variabile record definita dall'istruzione TYPE...END TYPE sono esplicitamente definiti dalle clausole AS, essi non vengono influenzati dalle istruzioni DEFtipo di QBasic.

L'aggiunta di un carattere di dichiarazione del tipo (%, &, !, # o $) al nome di una variabile consente di definire la variabile di un tipo specifico; ad esempio, l'aggiunta di una e commerciale (&) alla fine del nome della variabile la definisce di tipo intero lungo. Inoltre, se il tipo di una variabile viene dichiarato utilizzando una clausola AS, quella variabile viene definita in base al tipo specificato. Ad esempio, nell'istruzione:

```
DIM spazioLibero AS LONG
```

la variabile spazioLibero è un intero lungo. Questa istruzione esplicita ha la precedenza rispetto a qualsiasi valore predefinito creato mediante le istruzioni DEFtipo.

Suggerimenti

- Dato che sono di tipo diverso, i%, i&, i!, i# e i$ sono nomi di variabili distinti e ognuno di essi può contenere un valore differente. Tuttavia, dato che l'utilizzo di variabili diverse con nomi analoghi potrebbe risultare fuorviante, è bene evitarlo.
- Se DEFtipo viene utilizzata per impostare il tipo di una funzione definita da un'istruzione DEF FN, è necessario ricordare che il nome della funzione inizia immediatamente a destra delle lettere FN. Se ad esempio viene creata una funzione denominata FNMedia, è necessario utilizzare l'istruzione DEFDBL M per definire FNMedia come valore a precisione doppia.

Esempio

Il programma di seguito riportato utilizza le istruzioni DEFtipo per impostare il tipo di variabile predefinito per le variabili di ciclo, gli importi totali e le stringhe di richiesta.

```
DEFDBL A, G      'ammont, globTotale sono in precisione doppia
DEFINT I         'i è un intero
DEFLNG X         'è un intero lungo
DEFSNG S         'subTotale è in precisione singola
DEFSTR N         'Nosi è una stringa


CLS

'inizializza le variabili
globTotale = 0#
x = 1

DO
   PRINT "Quanti articoli sulla ricevuta"; x;  'ottiene il numero di articoli
                                               'sulla ricevuta
   INPUT articoliSuRicevuta

   subTotale = 0!                              'inizializza subTotale
   FOR i = 1 TO articoliSuRicevuta             'per ciascun articolo sulla
                                               ' ricevuta
        PRINT "Inserisci l'ammontare dell' articolo"; i;
        INPUT ; ammont                         'ottiene l'ammontare degli
                                               ' articoli
        subTotale = subTotale + ammont         'aggiunge l'ammontare degli
                                               'articoli a subTotale
        globTotale = globTotale + ammont       'aggiunge l'ammontare degli
                                               'articoli a globTotale
        PRINT TAB(70); USING "L. #######"; subTotale

   NEXT

   PRINT TAB(70); "----------"                 'stampa il totale globale
                                               'corrente
   PRINT TAB(70); USING "L. #######"; globTotale
   PRINT

   INPUT "Desideri aggiungere più ricevute"; NoSi 'chiede se l'utente
   x = x + 1                                   'vuole inserire più ricevute
LOOP UNTIL UCASE$(NoSi) = "N"                  'altrimenti abbandona il
                                               'programma
```

DOUBLE

Si consulti anche

COMMON, DECLARE, DEF FN, DIM, FUNCTION, INTEGER, LONG, REDIM, SHARED, SINGLE, STATIC, STRING, SUB, TYPE...END TYPE

Scopo

In una clausola AS, la parola chiave DOUBLE specifica il tipo di una variabile o di una funzione come valore a virgola mobile in precisione doppia; deve essere utilizzata quando è necessario gestire numeri grandi con precisione, come nei calcoli scientifici e in quelli commerciali.

Sintassi

```
var AS DOUBLE
```

var è il nome della variabile; la sintassi corretta dipende dall'istruzione in cui questa viene dichiarata.

Utilizzo

```
DIM totale AS DOUBLE
```

Definisce totale come una variabile a precisione doppia.

```
DECLARE FUNCTION RadiceCubica# (numero AS DOUBLE)
```

Dichiara una funzione che assume un argomento a precisione doppia.

Descrizione

La parola chiave DOUBLE può essere utilizzata in qualsiasi istruzione di QBasic che comprende una clausola AS. Nella Tabella 1.6 sono elencate le istruzioni di questo tipo. Le variabili dichiarate di tipo a precisione doppia possono contenere valori in *virgola mobile* (ovvero valori che presentano delle cifre dopo la virgola decimale), con una precisione fino alla quindicesima cifra decimale. Questo significa che, sebbene una variabile a precisione doppia possa contenere valori superiori, soltanto 15 cifre sono precise. Dato che le variabili a precisione doppia possono utilizzare la notazione esponenziale (cioè la notazione scientifica), esse possono contenere valori estremamente ampi: da 4.940656458412465D-324 a 1.797693134862315D+308per i numeri positivi, da -1.797693134862315D+308 a -4.940656458412465D-324 per i numeri negativi.

La notazione esponenziale per i numeri a precisione doppia viene espressa nel seguente modo:

±mantissaD±esponente

Un esempio di questo formato è il numero 4,107D+6, che equivale a $4{,}107 \times 10^6$ e a 4,107,000.

Tabella 1.6 Le istruzioni QBasic che utilizzano la clausola AS

COMMON	DIM	STATIC
DECLARE SUB	FUNCTION	SUB
DECLARE FUNCTION	REDIM	TYPE...END TYPE
DEF FN	SHARED	

Si noti che, secondo la definizione matematica della notazione scientifica, quando l'esponente è negativo il valore dell'espressione è frazionario, non negativo, come nella seguente espressione:

4.107D-6 = $1.107x10^{-6}$ = 0.000004107

Errori

Se a una variabile a precisione doppia viene assegnato un valore non compreso nell'intervallo corrispondente, QBasic visualizza il messaggio di errore: "Overflow".

Suggerimenti

Invece di dichiarare una variabile utilizzando la clausola AS DOUBLE, è possibile aggiungere il carattere di dichiarazione del tipo # alla fine del nome della variabile stessa. Tuttavia, è necessario tenere presente che il simbolo di numero (#) diventa parte del nome e quindi deve essere inserito ogniqualvolta la variabile viene utilizzata.

È possibile inoltre utilizzare un'istruzione DEFDBL per dichiarare che tutte le variabili il cui nome inizia con determinate lettere sono a precisione doppia (si faccia riferimento alla voce DEFtipo).

HEX$

Si consulti anche

OCT$

Scopo

La funzione HEX$ consente di convertire un'espressione di tipo intero o intero lungo in una stringa che contiene il valore esadecimale (base 16) corrispondente. Questa funzione può essere utilizzata quando occorre gestire routine in linguaggio assembler.

Sintassi

```
HEX$(espressionenumerica)
```

espressionenumerica è la variabile numerica o l'espressione convertita dalla funzione HEX$.

Utilizzo

```
PRINT HEX$(65536)
```

Visualizza il valore esadecimale corrispondente al numero decimale intero 65536, ovvero 10000.

Descrizione

Generalmente si preferisce utilizzare i numeri decimali (base 10), tuttavia altri sistemi, quali quello binario (base 2), ottale (base 8) ed esadecimale (base 16) risultano più utili nei processi di tipo digitale. QBasic fornisce la funzione HEX$ per convertire i numeri decimali in stringhe contenenti la loro rappresentazione in esadecimale. Il valore specificato è una variabile numerica o un'espressione di qualsiasi tipo numerico; QBasic lo converte in un intero o in un intero lungo. Si noti che qualsiasi parte frazionaria viene ignorata.

Errori

Se il valore specificato non rientra nell'intervallo da -2147483648 a 2147483647, QBasic visualizza il messaggio di errore: "Overflow".

Esempio

L'esempio di seguito riportato utilizza le funzioni PEEK e HEX$ per stampare la rappresentazione esadecimale del valore dello status dei tasti.

```
CLS
DEF SEG = 0              'accede all'area dati del BIOS

PRINT "Premi i tasti Maiusc, Ctrl, Alt,"
PRINT "      Bloc Num, Bloc Maiusc, Bloc Scorr."
PRINT "Premi Esc per terminare."

DO
    LOCATE 4, 1          'posizione del cursore

 'ottiene i dati all'indirizzo 0417 esadecimale, convertendolo in esadecimale
    PRINT RIGHT$("0" + HEX$(PEEK(&H417)), 2)
LOOP UNTIL INKEY$ = CHR$(27)
```

INTEGER

Si consulti anche

COMMON, DECLARE, DEF FN, DIM, DOUBLE, FUNCTION, LONG, REDIM, SHARED, SINGLE, STATIC, STRING, SUB, TYPE...END TYPE

Scopo

La parola chiave INTEGER consente di specificare in una clausola AS il tipo di una variabile o di una funzione come intero; deve essere utilizzata soltanto quando si tratta di numeri piccoli senza parti frazionarie (interi), per operazioni quali l'accesso alle coordinate dello schermo.

Sintassi

```
var AS INTEGER
```

var è il nome della variabile; la sintassi corretta dipende dall'istruzione in cui viene dichiarata.

Utilizzo

```
DIM anni AS INTEGER
```

Definisce anni come variabile di tipo intero.

```
DECLARE FUNCTION Cubo (numero AS INTEGER)
```

Dichiara una funzione che assume un argomento intero.

Descrizione

La parola chiave INTEGER può essere utilizzata in qualsiasi istruzione QBasic che comprenda una clausola AS.

Le variabili dichiarate come INTEGER possono contenere numeri interi compresi nell'intervallo da -32768 a 32767. Se è necessario utilizzare un intervallo più ampio di numeri interi, occorre avvalersi della parola chiave LONG.

Commenti

Le operazioni matematiche sulle variabili di tipo intero o intero lungo vengono eseguite più facilmente rispetto a quelle sulle variabili a precisione semplice o a precisione doppia. Le variabili dei primi due tipi possono essere direttamente rappresentate nei registri della CPU, mentre le tre parti che compongono le variabili a precisione semplice e quelle a precisione doppia (*segno*, *mantissa* ed *esponente*) devono essere calcolate, rallentando in questo modo le operazioni matematiche. Per questo motivo è preferibile utilizzare le variabili di tipo intero o intero lungo, quando possibile.

Errori

Se a una variabile di tipo intero viene assegnato un valore non compreso nell'intervallo corrispondente, QBasic visualizza il messaggio di errore: "Overflow".

Suggerimenti

Invece di dichiarare una variabile utilizzando la clausola AS INTEGER, è possibile aggiungere il carattere di dichiarazione del tipo % alla fine del suo nome. Tuttavia, è necessario tenere presente che il segno di percentuale (%) diventa parte del nome e quindi deve essere inserito ogniqualvolta la variabile viene utilizzata.

È inoltre possibile utilizzare l'istruzione DEFINT per dichiarare che tutte le variabili il cui nome inizia con determinate lettere sono intere (si faccia riferimento alla voce DEFtipo).

LET

Si consulti anche

LSET, TYPE...END TYPE

Scopo

L'istruzione LET consente di assegnare il valore di un'espressione a una variabile, a un elemento di array, oppure a una variabile o elemento definito dall'utente mediante TYPE...END TYPE. Questa istruzione deve essere utilizzata per inizializzare le variabili o per modificarne il valore.

Sintassi

```
[LET] var = espressione
```

var è il nome della variabile a cui viene assegnato il valore di espressione.
La parola chiave LET è facoltativa.

Utilizzo

```
LET a = 0
```

Assegna il valore 0 alla variabile a.

```
x = x +1
```

Aumenta la variabile x di 1.

```
articolo(5) = 4.8
```

Assegna il valore 4.8 al quinto elemento dell'array articolo.

```
articolo.numpart = "26-3004B"
```

Assegna la stringa "26-3004B" all'elemento numpart nella variabile record definita dall'utente articolo.

Descrizione

L'istruzione LET è l'operatore di assegnazione di QBasic, che consente di assegnare i valori alle variabili; può essere utilizzata per assegnare i valori iniziali delle variabili in un programma, oppure per modificarne i valori in un momento successivo. Ad esempio, i programmatori assegnano spesso i valori iniziali, quando iniziano a progettare un programma, nel modo seguente:

```
x = 0: y = 0: z = 0
```

QBasic consente di utilizzare la stessa variabile su entrambi i lati del segno di uguale, permettendo in questo modo di apportare le modifiche riguardanti il valore corrente della variabile; ad esempio, l'istruzione x = x + 1 aumenta di 1 il valore di x. Nell'istruzione y = 5*y, QBasic calcola l'espressione 5*y assegnando quindi il valore risultante a y.

Sia la variabile che l'espressione presenti nell'istruzione LET possono essere variabili record definite dall'istruzione TYPE...END TYPE; tuttavia entrambe devono essere dichiarate come tipo definito dall'utente. Ad esempio, le istruzioni DIM:

```
DIM nuovoArticolo AS Inventario
DIM articolo AS Inventario
```

dichiarano due variabili record dello stesso tipo, permettendo di eseguire l'istruzione di assegnazione di seguito riportata.

```
articolo = nuovoArticolo
```

Si osservi che tutto ciò funziona soltanto quando la clausola AS è identica per entrambe le variabili record; se sono diverse, è possibile utilizzare l'istruzione LSET per assegnare il contenuto di una variabile record all'altra.

Esempio

L'esempio di seguito riportato utilizza l'istruzione di assegnazione LET per creare una tabella di quadrati e radici quadrate dei numeri da 1 a 20.

```
CLS

PRINT "           Numero    Radice quadrata"
PRINT "Numero     Quadrato"
PRINT

FOR i% = 1 TO 20
    LET quadrato# = i% ^ 2               'assegna a quadrato# il quadrato di i
    LET radiceQuadrata# = SQR(i%)        'assegna a radiceQuadrata# la radice
                                         'quadrata di i
    PRINT USING "##        ####          #.#####"; i%; quadrato#; radiceQuadrata#
NEXT
```

LONG

Si consulti anche

COMMON, DECLARE, DEF FN, DIM, DOUBLE, FUNCTION, INTEGER, REDIM, SHARED, SINGLE, STATIC, STRING, SUB, TYPE...END TYPE

Scopo

La parola chiave LONG consente di specificare in una clausola AS il tipo di variabile o di funzione come intero lungo; deve essere utilizzata quando è necessario avvalersi di numeri molto grandi senza parti frazionarie (interi), ad esempio importi in lire.

Sintassi

```
var AS LONG
```

var è il nome della variabile; la sintassi esatta dipende dall'istruzione in cui viene dichiarata.

Utilizzo

```
DIM deficit AS LONG
```

Definisce deficit come variabile di tipo intero lungo.

```
DECLARE FUNCTION Cubo (numero AS LONG)
```

Dichiara una funzione che assume un argomento di tipo intero lungo.

Descrizione

La parola chiave LONG può essere utilizzata in qualsiasi istruzione QBasic che comprende una clausola AS.

Le variabili dichiarate come LONG possono contenere numeri interi compresi nell'intervallo da -2147483648 a 2147483647. Se è necessario utilizzare un intervallo più ampio, occorre avvalersi della parola chiave DOUBLE. Le variabili dichiarate come valori a precisione doppia possono contenere numeri a virgola mobile (ovvero valori che presentano delle cifre dopo la virgola decimale), anche se, ovviamente, è possibile utilizzare la variabile anche nel caso di numeri interi (si faccia riferimento alla voce DOUBLE).

Commenti

Le operazioni matematiche sulle variabili di tipo intero e intero lungo vengono eseguite più facilmente rispetto a quelle sulle variabili a precisione semplice o a precisione doppia. Le variabili dei primi due tipi possono essere direttamente rappresentate nei registri della CPU, mentre le tre parti che compongono le variabili a precisione semplice e quelle a precisione doppia (segno, mantissa ed esponente) devono essere calcolate, rallentando in questo modo le operazioni matematiche. Per questo motivo è preferibile utilizzare le variabili di tipo intero o intero lungo, quando possibile.

Errori

Se a una variabile di tipo intero lungo viene assegnato un valore non compreso nell'intervallo corrispondente, QBasic visualizza il messaggio di errore: "Overflow".

Suggerimenti

Invece di dichiarare una variabile utilizzando la clausola AS LONG, è possibile aggiungere il carattere di dichiarazione del tipo & alla fine del suo nome. Tuttavia, è necessario tenere presente che la e commerciale (&) diventa parte del nome e quindi deve essere inserita ogniqualvolta la variabile viene utilizzata.

È inoltre possibile utilizzare l'istruzione DEFLNG per dichiarare che tutte le variabili il cui nome inizia con determinate lettere sono di tipo intero lungo (si faccia riferimento alla voce DEFtipo).

OCT$

Si consulti anche

HEX$

Scopo

La funzione OCT$ consente di convertire un'espressione di tipo intero o intero lungo in una stringa che contiene il valore ottale (base 8) corrispondente. Questa funzione può essere utilizzata quando occorre gestire routine in linguaggio assembler.

Sintassi

```
OCT$(espressionenumerica)
```

espressionenumerica è la variabile numerica o l'espressione convertita dalla funzione OCT$.

Utilizzo

```
PRINT OCT$(511)
```

Visualizza il valore ottale corrispondente al numero decimale intero 511, ovvero 777.

Descrizione

Generalmente si preferisce utilizzare i numeri decimali (base 10), tuttavia altri sistemi di numerazione, quali quello binario (base 2), ottale (base 8) ed esadecimale (base 16) risultano più utili nei processi di tipo digitale. QBasic fornisce la funzione OCT$ per convertire i numeri decimali in stringhe contenenti la loro rappresentazione in ottale.

Il valore specificato è una variabile numerica o un'espressione di qualsiasi tipo numerico; QBasic lo converte in un intero o in un intero lungo. Si noti che qualsiasi parte frazionaria viene ignorata.

Errori

Se il valore specificato non rientra nell'intervallo da -2147483648 a 2147483647, QBasic visualizza il messaggio di errore: "Overflow".

Esempio

L'esempio di seguito riportato utilizza la funzione OCT$ per stampare la rappresentazione ottale dei numeri decimali immessi dall'utente.

```
CLS

PRINT "Per terminare, premi Invio o inserisci 0."
PRINT

DO
    INPUT "Inserisci un numero intero decimale"; dec%
    IF dec% <= 0 THEN EXIT DO
```

```
    PRINT dec%; "decimale corrisponde a "; OCT$(dec%); " ottale"
    PRINT
LOOP UNTIL dec% = 0
```

SINGLE

Si consulti anche

COMMON, DECLARE, DEF FN, DIM, DOUBLE, FUNCTION, INTEGER, LONG, REDIM, SHARED, STATIC, STRING, SUB, TYPE...END TYPE

Scopo

In una clausola AS, la parola chiave SINGLE specifica il tipo di una variabile o di una funzione come valore a virgola mobile in precisione semplice; deve essere utilizzata quando è necessario gestire numeri decimali.

Sintassi

```
var AS SINGLE
```

var è il nome della variabile; la sintassi corretta dipende dall'istruzione in cui viene dichiarata.

Utilizzo

```
DIM totale AS SINGLE
```

Definisce totale come una variabile a precisione semplice.

```
DECLARE FUNCTION RadiceCubica# (numero AS SINGLE)
```

Dichiara una funzione che assume un argomento a precisione semplice.

Descrizione

La parola chiave SINGLE può essere utilizzata in qualsiasi istruzione QBasic che comprenda una clausola AS.

Le variabili dichiarate di tipo a precisione semplice possono contenere valori in virgola mobile (ovvero valori che presentano delle cifre dopo il punto decimale), con una precisione fino alla settima cifra decimale. Tutto ciò significa che, sebbene una variabile a precisione semplice possa contenere valori superiori, soltanto 7 cifre sono precise. Dato che le variabili a precisione semplice possono utilizzare la notazione esponenziale (ovvero la notazione scientifica), esse possono contenere valori estremamente grandi: da 1.401298E-45 a 3.402823E+38 per i numeri positivi e da - 3.402823E+38 a -1.401298E-45 per i numeri negativi.

Un esempio di notazione esponenziale è il numero 4.107E+4, che equivale a 4.107×10^4 e a 41,070.

Si noti che, secondo la definizione matematica di notazione scientifica, quando l'esponente è negativo il valore dell'espressione è frazionario, non negativo, come nella seguente espressione:

4.107E-4 = 4.107x10-4 = 0.0004107

Errori

Se a una variabile a precisione semplice viene assegnato un valore non compreso nell'intervallo del tipo corrispondente, QBasic visualizza il messaggio di errore: "Overflow".

Suggerimenti

Invece di dichiarare una variabile utilizzando la clausola AS SINGLE, è possibile aggiungere il carattere di dichiarazione del tipo ! alla fine del suo nome. Tuttavia, è necessario tenere presente che il punto esclamativo (!) diventa parte del nome e quindi deve essere inserito ogniqualvolta la variabile viene utilizzata.

È possibile inoltre utilizzare un'istruzione DEFSNG per dichiarare che tutte le variabili il cui nome inizia con determinate lettere sono a precisione singola (si faccia riferimento alla voce DEFtipo).

È necessario tenere presente che in QBasic il tipo di variabile predefinita è a precisione semplice; questo valore può essere modificato tramite le istruzioni DEFtipo.

STRING

Si consulti anche

COMMON, DECLARE, DEF FN, DEFtipo, DIM, FIELD, FUNCTION, REDIM, SHARED, STATIC, STR$, SUB, TYPE...END TYPE.

Scopo

La parola chiave STRING consente di specificare in una clausola AS il tipo di una variabile o di una funzione come stringa. Una variabile di tipo stringa contiene una serie di caratteri. Questa parola chiave deve essere utilizzata quando è necessario gestire stringhe di caratteri, per operazioni quali la lettura del nome dell'utente.

Sintassi

```
var AS STRING
```

var è il nome della variabile; la sintassi corretta dipende dall'istruzione in cui viene dichiarata.

Utilizzo

```
DIM nome AS INTEGER
```

Definisce nome come variabile stringa.

```
DECLARE FUNCTION TastiIn$ (tasto AS STRING)
```

Dichiara una funzione che assume un argomento stringa e restituisce un valore stringa.

Descrizione

La parola chiave STRING può essere utilizzata in qualsiasi istruzione QBasic che comprenda una clausola AS. Le variabili dichiarate come STRING possono contenere qualunque dei caratteri ASCII o dei 128 caratteri del set di caratteri esteso IBM.

Errori

Se a una variabile stringa viene assegnata una stringa che contiene più di 32767 caratteri, QBasic visualizza il messaggio: "Overflow".

Se si cerca di assegnare un valore numerico a una variabile stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Suggerimenti

Invece di dichiarare una variabile utilizzando la clausola AS STRING, è possibile aggiungere il carattere di dichiarazione del tipo $ alla fine del suo nome. Tuttavia, è necessario tenere presente che il segno di dollaro ($) diventa parte del nome e quindi deve essere inserito ogniqualvolta la variabile viene utilizzata.

È possibile inoltre utilizzare un'istruzione DEFSTR per dichiarare che tutte le variabili il cui nome inizia con determinate lettere sono di tipo stringa (si faccia riferimento alla voce DEFtipo).

SWAP

Si consulti anche

TYPE...END TYPE

Scopo

L'istruzione SWAP scambia i valori di due variabili, di elementi di array, di variabili record definite da un'istruzione TYPE...END TYPE oppure di elementi di una variabile record. Questa istruzione deve essere utilizzata per scambiare velocemente i valori tra le variabili, ad esempio durante un ordinamento.

Sintassi

```
SWAP var1, var2
```

var1 e var2 sono le variabili che contengono i valori scambiati dall'istruzione SWAP.

Utilizzo

```
SWAP globTotale, sottoTotale
```

Scambia i valori di globTotale e sottototale.

Descrizione

L'istruzione SWAP di QBasic riunisce un'operazione suddivisa in tre fasi in una fase singola. Ad esempio:

```
SWAP globTotale, sottoTotale
```

equivale alle seguenti istruzioni:

```
temp = sottoTotale
sottoTotale = globTotale
globTotale = temp
```

Si noti che l'ultima operazione richiede l'utilizzo di una variabile temporanea.

Le due variabili possono essere semplici variabili, elementi di array, variabili record definite dall'istruzione TYPE...END TYPE oppure elementi di una variabile record, ma devono essere dello stesso tipo; in caso contrario QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

L'esempio di seguito riportato utilizza l'istruzione SWAP per eseguire un *bubble sort* su un array di tipo intero a 10 elementi. Il termine bubble sort (ovvero "ordinamento a palloncino"), indica un ordinamento i cui valori si elevano fino alla posizione corretta, proprio come i palloncini fluttuano verso la superficie. È uno degli algoritmi di ordinamento più semplice, ma è pittosto lento.

```
CLS

DEFINT A-Z

CONST FALSO = 0, VERO = NOT FALSO       'definisce VERO e FALSO

DIM array(1 TO 10) AS INTEGER           'dichiara un array di 10 elementi

PRINT "Inserisci 10 numeri interi, premi Invio dopo ciascun numero."
PRINT

FOR i = 1 TO 10
    INPUT array(i)                      'assegna i numeri interi agli elementi
                                        'dell'array
NEXT
PRINT

'realizza un ordinamento di tipo Bubble sort
DO
    interruttore = FALSO
    FOR i = 1 TO 9
        'controlla se due numeri adiacenti non sono in ordine
        IF array(i) > array(i + 1) THEN
            SWAP array(i), array(i + 1)        'se non in ordine, scambia
            interruttore = VERO
        END IF
    NEXT
LOOP WHILE interruttore

PRINT : PRINT "Eccoli ordinati:": PRINT

FOR i = 1 TO 10
    PRINT array(i);
NEXT
```

TYPE...END TYPE

Si consulti anche

COMMON, DECLARE, DEF FN, DIM, FUNCTION, REDIM, SHARED, STATIC, SUB, SWAP

Scopo

L'istruzione TYPE...END TYPE consente di creare un tipo di dati (a cui talvolta si fa riferimento come tipo definito dall'utente), che rappresenta una struttura consistente di una o più variabili. In questo modo è possibile semplificare la gestione dei dati. Ad esempio, è possibile creare una variabile denominata inventario che contiene diverse variabili legate ai dati dell'inventario.

Sintassi

```
TYPE tipoutente
   nomeelemento AS nometipo
   [nomeelememto AS nometipo]
   .
   .
END TYPE
```

tipoutente è il nome del tipo.
nomeelemento è il nome della variabile che è del tipo tipo.

Utilizzo

```
TYPE Degenti
   nome AS STRING * 15
   cognome AS STRING * 20
   DataNascita AS STRING * 6
END TYPE
```

Crea un tipo definito dall'utente, denominato Degenti.

```
TYPE Famiglia
   indirizzo AS STRING * 30
   Degente1 AS Degenti
   Degente2 AS Degenti
   Degente3 AS Degenti
END TYPE
```

Crea un tipo definito dall'utente, denominato Famiglia, che contiene gli elementi appartenenti al tipo Degenti.

Descrizione

Quasi tutti i linguaggi di programmazione consentono di creare nuovi tipi di dati; generalmente, ne viene creato uno che permette a una singola variabile di contenere contemporaneamente diverse variabili. Ad esempio, sviluppando un programma per la gestione delle polizze assicurative da parte di medici, è opportuno inserire una singola variabile contenente il nome del degente, l'indirizzo, il numero di assistenza sanitaria e così via.

Quando un tipo viene definito mediante l'istruzione TYPE...END TYPE, è necessario innanzitutto assegnare un nome a cui fare riferimento quando successivamente le variabili vengono definite con questo nuovo tipo. Il nome del tipo deve essere un identificatore valido di QBasic e non deve essere utilizzato da un'altra variabile.

È possibile definire ogni elemento del tipo definito dall'utente. Il nome dell'elemento può essere lo stesso di un'altra variabile, di un tipo definito dall'utente, di una funzione, di un elemento, oppure il nome di un sottoprogramma, ma non può essere il nome di un altro elemento nella stessa istruzione TYPE...END TYPE; inoltre non deve essere il nome di un array e non può contenere punti.

A questo punto è possibile assegnare un tipo a ogni elemento. Questo tipo può essere la parola chiave DOUBLE, INTEGER, LONG o SINGLE, un tipo di stringa a lunghezza fissa (STRING*x) oppure il nome di un altro tipo definito dall'utente, ma non può essere una stringa standard a lunghezza variabile.

È possibile definire un tipo particolare soltanto una volta in un modulo e non è possibile utilizzare le istruzioni TYPE...END TYPE nei sottoprogrammi o nelle funzioni; il metodo migliore per inserire queste istruzioni è quello di porle all'inizio di un modulo.

La definizione di un tipo definito dall'utente non crea alcuna variabile; la creazione di variabili di quel tipo (denominate variabili record) avviene utilizzando le istruzioni COMMON, DIM, REDIM, SHARED o STATIC. È inoltre possibile utilizzare la clausola AS nell'istruzione DECLARE SUB, DECLARE FUNCTION, DEF FN, FUNCTION o SUB per indicare che la variabile che si sta passando è di un tipo record.

Nella Figura 1.3 è riportato un esempio di un'istruzione TYPE...END TYPE e di una dichiarazione di variabile record.

Per accedere a un elemento in una variabile record è necessario utilizzare una sintassi speciale, specificando il nome della variabile record seguito da un punto (.) e poi il nome dell'elemento. Ad esempio, nella variabile record nascita, che è dichiarata nella Figura 1.3, si dovrebbe utilizzare la seguente istruzione per impostare il valore dell'elemento mese a 9:

```
nascita.mese = 9
```

Per utilizzare un elemento di una variabile record nella condizione di un'istruzione IF, è necessario utilizzare la stessa sintassi, ovvero:

```
IF 1992 - nascita.anno >= 18 THEN
   PRINT "Può votare"
END IF
```

Commenti

Sebbene non sia possibile utilizzare array nell'istruzione TYPE...END TYPE, è possibile creare un array che contenga i record definiti da questa.

Errori

Se viene creato un nome di un elemento contenente punti, QBasic visualizza il messaggio: "Un identificatore non può includere un punto".

```
TYPE Data ─────────────────Data è il nome del tipo

   mese AS INTEGER ──────┐   mese, giorno, anno sono i nomi degli elementi
   giorno AS INTEGER     ├── INTEGER è il tipo degli elementi
   anno AS INTEGER ──────┘

END TYPE

DIM nascita AS Data ───────nascita è una variabile record di tipo Data
```

Figura 1.3 Un esempio dell'istruzione TYPE...END TYPE e della dichiarazione di una variabile record

Esempio

L'esempio di seguito riportato utilizza l'istruzione TYPE...END TYPE per creare un tipo record che preleva le informazioni relative ai pianeti e le inserisce in una variabile singola. Il programma può essere sfruttato per scrivere e leggere le informazioni dal file di dati SOL.DAT.

```
TYPE Pianeti                    'tipo di dati contenente le informazioni sul
                                'pianeta
    nome AS STRING * 20         'nome del pianeta
    diametro AS LONG            'diametro del pianeta
    perielio AS DOUBLE          'punto dell'orbita più vicino al sole
    afelio AS DOUBLE            'punto dell'orbita più lontano dal sole
    maxTemp AS SINGLE           'temperatura massima della superficie
    minTemp AS SINGLE           'temperatura minima della superficie
    annoScoperta AS INTEGER     'anno di scoperta del pianeta
END TYPE

DIM pianeta AS Pianeti          'definisce la variabile pianeta

CLS
OPEN "SOL.DAT" FOR RANDOM AS #1 LEN = 50        'apri il file di dati

DO
    PRINT : PRINT : PRINT
    INPUT "Su quale pianeta si desiderano informazioni [1-9]"; numPianeta%
    'ottiene il numero

    'l'introduzione di un numero non compreso nell'intervallo o la pressione
    'di Invio, provoca l'uscita dal ciclo
    IF numPianeta% < 1 OR numPianeta% > 9 THEN EXIT DO

    GET #1, numPianeta%, pianeta                'ottieni i dati dal file

    PRINT : PRINT "Informazioni sul pianeta "; pianeta.nome 'stampa il titolo
    PRINT STRING$(79, 45): PRINT

    'se il record è vuoto le informazioni saranno richieste all'utente
    IF pianeta.nome = STRING$(20, 0) THEN
        INPUT "Nome"; pianeta.nome
        INPUT "Diametro (km)"; pianeta.diametro
        INPUT "Perielio (km)"; pianeta.perielio
        INPUT "Afelio (km)"; pianeta.afelio
        INPUT "Temperatura massima (Celsius)"; pianeta.maxTemp
        INPUT "Temperatura minima (Celsius)"; pianeta.minTemp
        INPUT "Scoperto nel"; pianeta.annoScoperta
```

```
        'scrive le informazioni sul file
        PUT #1, numPianeta%, pianeta

        PRINT : PRINT "Informazioni registrate": PRINT : PRINT

'se il record non è vuoto le informazioni contenute sono visualizzate sul
'video
    ELSE
        PRINT "Il diametro è"; pianeta.diametro; "chilometri"
        PRINT "Il perielio è"; pianeta.perielio; "chilometri"
        PRINT "L'afelio è"; pianeta.afelio; "chilometri"
        PRINT "La temperatura massima è"; pianeta.maxTemp; "gradi"
        PRINT "La temperatura minima è"; pianeta.minTemp; "gradi"
        PRINT "Scoperto nel"; pianeta.annoScoperta
    END IF
LOOP            'torna all'inizio del ciclo

CLOSE
```

Capitolo 2

Controllo del flusso

INTRODUZIONE

Solo pochi programmi eseguono le istruzioni dalla prima all'ultima in una sequenza continua; generalmente, un programma esegue compiti diversi in base ai dati che genera oppure alle istruzioni che riceve dall'utente. Un programma progettato in modo ottimale contiene una sezione separata del codice per ogni compito e l'esecuzione salta alla sezione corretta nel punto in cui deve essere eseguita l'operazione associata.

Spesso è necessario eseguire ripetutamente una singola operazione; ad esempio, un programma potrebbe dover eseguire una ricerca e sostituzione completa su un romanzo lungo 100 pagine, sostituendo ogni occorrenza del nome della protagonista "Beatrice" con "Ilaria". Un altro potrebbe dover stampare tutti gli assegni delle buste paghe dei dipendenti di una società. In questi casi i computer risultano particolarmente utili, dato che eseguono continuamente la stessa sequenza di istruzioni senza lamentarsi, effettuando quindi quei lavori ripetitivi e noiosi che non sono graditi agli esseri umani.

In questo capitolo vengono esaminati i metodi in base ai quali QBasic consente di controllare la sequenza di esecuzione delle istruzioni. Nella Tabella 2.1 sono riportate le istruzioni di controllo del flusso in ordine alfabetico.

SALTO

Se un programma comprende diverse sezioni, ognuna delle quali esegue un compito particolare, deve essere in grado di trasferire il controllo da una sezione all'altra. In QBasic, uno dei metodi che permette di eseguire questa operazione consiste nell'utilizzare una istruzione che esegue un *salto*; un *salto incondizionato* trasferisce sempre il controllo alla sezione indicata, mentre un *salto condizionato* lo trasferisce soltanto se viene soddisfatta una determinata condizione.

Tabella 2.1 Le istruzioni di controllo del flusso riportate in ordine alfabetico

Istruzione	Descrizione
DO...LOOP	Ripete un gruppo di istruzioni finché una determinata condizione è vera oppure fino a quando non lo diventa
EXIT	Trasferisce il controllo da un sottoprogramma, da una funzione definita dall'utente oppure da una struttura di ciclo prima che le normali condizioni di uscita siano soddisfatte
FOR NEXT	Ripete un gruppo di istruzioni per un determinato numero di volte
GOTO	Esegue un salto incondizionato alla riga specificata
ON...GOSUB	Salta a una subroutine in una delle righe specificate. Al completamento della subroutine, il controllo ritorna all'istruzione che segue l'istruzione ON...GOSUB
ON...GOTO	Salta a una delle diverse righe specificate
WHILE...WEND	Ripete un blocco di istruzioni finché una determinata condizione logica è vera

Salto incondizionato

Il metodo più semplice per indirizzare il flusso di esecuzione a un'altra parte del programma consiste nell'utilizzo di un salto incondizionato, realizzabile mediante l'istruzione GOTO, come nel seguente esempio:

```
GOTO Termine
```

che trasferisce il controllo alla riga che contiene l'etichetta Termine; questa istruzione può essere utilizzata anche con un numero di riga (nel caso il programma ne faccia uso). La riga di destinazione deve trovarsi nella stessa parte di programma dell'istruzione GOTO, dato che non è possibile sfruttare questa istruzione per saltare a un sottoprogramma o a una funzione definita dall'utente diversi.

GOTO si differenza da una chiamata di funzione: dopo il salto, il programma esegue le istruzioni successive alla riga di destinazione fino a quando non incontra un'altra istruzione di salto o fino al termine del programma. L'esecuzione del programma non ritorna alla riga successiva all'istruzione GOTO, come avviene con GOSUB (si consulti il Capitolo 4). L'eccezione a questa regola è rappresentata dall'utilizzo di GOTO con l'istruzione ON ERROR GOTO per definire una routine di gestione degli errori. In questo caso, quando si incontra un'istruzione RESUME successiva il controllo ritorna alla sezione del programma da cui è stato effettuato il salto.

Salto condizionato

Sebbene di per se stessa l'istruzione GOTO sia incondizionata, può essere utilizzata come parte di un'istruzione condizionata. Ad esempio, le istruzioni di seguito riportate provano il salto alla riga 100 soltanto quando la prima lettera contenuta nella variabile stringa risposta$ è S.

```
240 INPUT "Prospetto da stampare"; risposta$
250 IF UCASE$(LEFT$(risposta$, 1)) = "S" THEN 1000
```

Si noti che la parola chiave GOTO è implicita quando si utilizza l'istruzione IF...THEN. Tuttavia, se l'istruzione di destinazione ha un'etichetta invece di un numero di riga, è indispensabile inserire la parola chiave GOTO, come riportato nel seguente esempio:

```
250 IF UCASE$(LEFT$(risposta$, 1)) = "S" THEN GOTO Stampa
```

Anche un'altra versione dell'istruzione GOTO è condizionata: l'istruzione ON...GOTO può dirigere il programma in modo che salti a una delle diverse righe indicate; il valore di una variabile o di un'espressione determina quale. Nell'istruzione:

```
900 ON selezione% GOTO 1000, 2000, 3000, Inizio, Fine
```

il controllo salta a uno dei cinque numeri di riga o etichette a seconda del valore della variabile di tipo intero selezione%. Se quest'ultima contiene il valore 1 il programma salta alla riga 1000, se invece contiene il valore 2, il programma salta alla riga 2000 e così via. Se selezione% contiene il valore 0, oppure, in questo caso, un valore superiore a 5, non si verifica alcun salto e il controllo passa all'istruzione successiva alla riga 900. È sempre necessario assicurarsi che la variabile selezione% abbia valore 0 o superiore; se la variabile è negativa, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita".

Sottoprogrammi

L'istruzione ON...GOSUB è analoga a ON...GOTO, tranne per il fatto che dopo l'esecuzione del sottoprogramma il controllo ritorna all'istruzione che segue ON...GOSUB.

Entrambe queste istruzioni vengono fornite soltanto per mantenere la compatibilità con le versioni precedenti di Basic, infatti in QBasic il metodo preferito per l'esecuzione condizionata di gruppi di istruzioni consiste nell'utilizzo dell'istruzione SELECT CASE (si consulti il Capitolo 3). Questa istruzione ha il vantaggio di essere più flessibile e di non avere le limitazioni presenti invece in ON...GOTO e ON...GOSUB.

CICLI

Un *ciclo* è una sequenza di istruzioni che un programma esegue più di una volta in successione. QBasic dispone di due tipi di cicli: il primo, controllato dall'istruzione FOR...NEXT, è un ciclo che si ripete un numero fisso di volte (il programma specifica il numero di ripetizioni quando inizia il ciclo); nel secondo tipo, il numero effettivo di iterazioni non viene specificato in anticipo, ma viene determinato dal risultato di un test logico effettuato durante ogni passaggio del ciclo. L'esecuzione prosegue finché non viene soddisfatta una determinata condizione.

L'istruzione FOR...NEXT

Nel ciclo FOR...NEXT, QBasic ripete le istruzioni presenti tra le parole chiave FOR e NEXT per un numero di volte determinato mediante l'assegnazione di un intervallo di valori alla variabile il contatore di ciclo; ad esempio:

```
FOR contatore% = 1 TO 10
   PRINT contatore%;
NEXT contatore%
```

imposta il valore iniziale del contatore a 1 e quindi esegue il ciclo. Il contatore viene incrementato a ogni ripetizione fino a quando diventa superiore al valore massimo, che in questo caso è 10. Dato che la singola istruzione presente nel corpo del ciclo stampa soltanto il valore del contatore, l'output del programma è una serie di numeri da 1 a 10. QBasic consente di utilizzare la clausola STEP per specificare il valore dell'incremento del ciclo FOR...NEXT, che per default è 1. Ad esempio, il ciclo di seguito riportato conta fino a 10 a due alla volta:

```
FOR contatore% = 1 TO 10 STEP 2
   PRINT contatore%;
NEXT contatore%
```

generando l'output:

```
1 3 5 7 9
```

La clausola STEP può anche essere utilizzata per diminuire il contatore di ciclo; il seguente programma conta all'indietro tre alla volta:

```
FOR contatore% = 10 TO 1 STEP -3
   PRINT contatore%;
NEXT contatore%
```

generando l'output:

```
10 7 4 1
```

Nel corpo del ciclo è possibile utilizzare il numero di istruzioni desiderato; l'intera sequenza viene ripetuta fino a quando il valore del contatore si trova al di fuori dell'intervallo specificato, a questo punto il programma continua con l'istruzione successiva alla parola chiave NEXT.

L'istruzione WHILE...WEND

L'istruzione WHILE...WEND può essere utilizzata nelle situazioni in cui il programma deve eseguire una serie di istruzioni per un numero indeterminato di volte. Quando QBasic incontra per la prima volta la parola chiave WHILE, verifica il valore dell'espressione logica o della variabile associata e, se risulta vero (non zero), esegue le istruzioni successive. Quando il programma individua la parola chiave corrispondente WEND, restituisce il controllo alla parte superiore del ciclo e verifica nuovamente il valore dell'espressione associata o della variabile: se è ancora vero, il processo viene ripetuto; se è falso (zero) l'esecuzione del programma continua con l'istruzione che

segue la parola chiave WEND (si consulti il Capitolo 3 per una spiegazione relativa a ciò che rende un'espressione logicamente vera o falsa). Ad esempio, il programma di seguito riportato:

```
contatore% = 1
WHILE contatore% <=10
   PRINT contatore%;
   contatore% = contatore% + 1
WEND
```

genera l'output:

```
1 2 3 4 5 6 7 8 9 10
```

Questo tipo di ciclo viene definito un *ciclo con test iniziale.* Se l'espressione è già falsa quando si incontra per la prima volta la parola chiave WHILE, (ad esempio se contatore% è inizialmente 11), il programma ignora completamente il ciclo e non viene eseguita alcuna istruzione al suo interno.

Dato che le variabili numeriche e le funzioni di QBasic restituiscono valori che possono essere interpretati logicamente e numericamente, l'istruzione WHILE...WEND presenta diverse applicazioni. L'esempio di seguito riportato apre un file ad accesso sequenziale avente una lunghezza non specificata e utilizza l'istruzione WHILE...WEND per leggerlo fino a quando non ne incontra la fine.

```
OPEN file$ FOR INPUT AS #1
WHILE NOT EOF(1)
   LINE INPUT #1, record$
   PRINT record$
WEND
```

L'istruzione DO...LOOP

L'istruzione DO...LOOP è più potente e flessibile rispetto a WHILE...WEND. L'esempio precedente può essere facilmente riscritto utilizzando questa istruzione:

```
OPEN file$ FOR INPUT AS #1
DO WHILE NOT EOF(1)
   LINE INPUT #1, record$
   PRINT record$
LOOP
```

Questa versione è quasi identica a quella precedente, compreso il test elementare per stabilire la fine del file. Oltre a ripetere una serie di istruzioni finché una condizione è vera, un'istruzione DO...LOOP consente di ripetere le istruzioni soltanto finché una condizione è falsa, continuando a eseguire il ciclo fino a quando la condizione non diventa vera. Ciò si ottiene utilizzando la parola chiave UNTIL, come riportato nel seguente esempio:

```
OPEN file$ FOR INPUT AS #1
DO UNTIL EOF(1)
   LINE INPUT #1, record$
   PRINT record$
LOOP
```

Entrambe le istruzioni DO...LOOP dei due esempi precedenti sono, come le istruzioni WHILE...WEND, cicli con test iniziale; questo significa che, se il test fallisce quando il ciclo è specificato per la prima volta (ad esempio, se il file è vuoto), le istruzioni all'interno del ciclo non vengono mai eseguite. Tuttavia, un ciclo creato da un'istruzione DO...LOOP può anche essere un ciclo con test finale, in cui il corpo del ciclo viene sempre eseguito almeno una volta; ad esempio il ciclo:

```
DO
   PRINT SPACE$(81 - POS(0))
LOOP WHILE CSRLIN < 23
```

cancella sempre tutto il contenuto della riga corrente, anche nel caso in cui il cursore si trovi già in corrispondenza della riga 23 del video o più sotto. È inoltre possibile utilizzare UNTIL invece di WHILE in un ciclo con test finale, come riportato nel seguente esempio:

```
PRINT "Stampante pronta, premere P per continuare"
DO
   pronta$ = UCASE$(INPUT$(1))
LOOP UNTIL pronta$ = "P"
```

Gli esempi precedenti dimostrano che l'istruzione DO...LOOP è molto più versatile rispetto all'istruzione WHILE...WEND. Questa è la ragione per cui è diventata il metodo preferito per la realizzazioni di cicli che comportano test logici.

Interruzione dei cicli

QBasic consente di uscire da un ciclo in corrispondenza di un'istruzione posta al suo interno, oltre che al punto normale di uscita, persino se la normale condizione di uscita non è stata soddisfatta. L'istruzione EXIT DO interrompe l'esecuzione del ciclo DO...LOOP da un punto qualsiasi posto tra le parole chiavi DO e LOOP e riprende l'esecuzione in corrispondenza dell'istruzione successiva alla parola chiave LOOP. L'istruzione EXIT FOR esegue la stessa operazione in un ciclo FOR...NEXT.

La seguente serie di istruzioni utilizza un ciclo FOR...NEXT per ricercare un carattere impostato in una posizione casuale in una stringa di spazi vuoti. Se il carattere viene individuato, il programma utilizza l'istruzione EXIT FOR per uscire dal ciclo; il contatore i% indica la posizione del carattere.

```
RANDOMIZE TIMER

a$ = SPACE$(10000)
x% = INT(RND * 10000) + 1
MID$(a$, x%, 1) = CHR$(INT(RND * 223) + 32)

FOR i% = 1 TO 10000
   IF MID$(a$, i%, 1) <> " " THEN EXIT FOR
NEXT i%
IF i% > 10000 THEN
   PRINT "Non individuato!"
ELSE
   PRINT i%; MID$(a$, i%, 1)
END IF
```

QBasic fornisce istruzioni EXIT analoghe per uscire dalle funzioni definite dall'utente e dai sottoprogrammi. Ad esempio, EXIT DEF consente di uscire da una chiamata di funzione DEF FN prima che venga eseguita l'istruzione END DEF. EXIT SUB ed EXIT FUNCTION eseguono lo stesso compito per i sottoprogrammi e le funzioni definite dall'utente. Non esiste alcuna istruzione di uscita equivalente per l'istruzione WHILE...WEND.

DO...LOOP

Si consulti anche

EXIT, FOR...NEXT, WHILE...WEND

Scopo

L'istruzione DO...LOOP consente di ripetere l'esecuzione di un gruppo di istruzioni finché una condizione iniziale o finale è vera, oppure fino a quando non lo diventa. Questa istruzione deve essere utilizzata per operazioni di conteggio, iterazione o per ripetere una data procedura in base a una determinata condizione.

Sintassi

L'istruzione DO...LOOP ha due possibili sintassi; la scelta tra le due dipende dall'effettuazione del test d'uscita all'inizio o alla fine del ciclo.

Per eseguire il test all'inizio del ciclo, è necessario utilizzare la seguente sintassi:

```
DO [{WHILE | UNTIL} condizione]
   [bloccoistruzioni]
LOOP
```

Se il test d'uscita è già vero quando la condizione viene valutata per la prima volta, il ciclo non viene eseguito. Per eseguire il test alla fine del ciclo, è sufficiente utilizzare la seguente sintassi:

```
DO
   [bloccoistruzioni]
LOOP [{WHILE | UNTIL} condizione]
```

Il ciclo viene sempre eseguito almeno una volta, a prescindere dal fatto che la condizione d'uscita sia già vera nel momento in cui il test viene eseguito la prima volta.

bloccoistruzioni rappresenta le istruzioni che vengono ripetute; queste sono facoltative, i cicli vuoti che non eseguono alcuna operazione fino a quando non si verifica un evento esterno (ad esempio la pressione di un tasto) sono permessi.

condizione è una qualsiasi espressione logica; deve essere valutata o vera (un valore diverso da zero) falsa (il valore zero). Se si utilizza la parola chiave WHILE, il ciclo viene ripetuto finché l'espressione rimane vera. Utilizzando UNTIL invece, il ciclo viene ripetuto fino a quando l'espressione non diventa vera.

Utilizzo

```
DO
LOOP WHILE INKEY$ = ""
```

Sospende l'esecuzione di un programma fino a quando non viene premuto un tasto.

```
DO UNTIL PLAY(1) = 0
LOOP
```

Sospende l'esecuzione di un programma fino a quando il buffer musicale di sottofondo non diventa vuoto.

```
DO
   x% = INT(RND * 80) + 1
   y% = INT(RND * 24) + 1
   LOCATE y%, x%
   PRINT "*"
LOOP UNTIL INKEY$ = CHR$(27)
```

Stampa degli asterischi in posizioni casuali sullo schermo finché non viene premuto il tasto ESC.

Descrizione

L'istruzione DO...LOOP permette di eseguire un ciclo finché una determinata condizione è vera oppure fino a quando non lo diventa. Questa istruzione consente inoltre di verificare la condizione d'uscita all'inizio o alla fine del ciclo. Se il test viene inserito all'inizio del ciclo e se la condizione d'uscita è già stata soddisfatta, il ciclo non viene eseguito; se invece il test viene inserito alla fine del ciclo, il corpo del ciclo viene sempre eseguito almeno una volta, anche se la condizione d'uscita è già vera.

È necessario assicurarsi che qualsiasi ciclo creato sia in grado di terminare. Se viene realizzato un ciclo in cui la condizione d'uscita non viene mai soddisfatta, il programma viene eseguito all'infinito.

L'istruzione EXIT DO può essere utilizzata a metà di un ciclo per uscirne prima che la condizione d'uscita diventi vera; trasferisce il controllo alla prima istruzione presente dopo la parola chiave LOOP. Sebbene QBasic restituisca -1 per rappresentare un'espressione vera, in pratica qualsiasi valore diverso da zero viene valutato come vero, quindi è possibile utilizzare qualsiasi variabile numerica o funzione come espressione di controllo in un'istruzione DO...LOOP.

Errori

Qualsiasi istruzione DO deve essere seguita da una corrispondente istruzione LOOP, proprio come ogni LOOP deve essere preceduta da un DO. Se il programma esegue un'istruzione LOOP senza avere individuato un DO precedente, QBasic visualizza il messaggio di errore: "LOOP senza DO". Al contrario, a un'istruzione DO priva della successiva LOOP corrisponde il messaggio di errore: "DO senza LOOP".

Avvertenze

Sebbene l'istruzione GOTO non produca immediatamente un errore, è meglio non utilizzarla per uscire da DO...LOOP; se per uscire da un ciclo viene ripetutamente utilizzata l'istruzione GOTO, lo stack del programma cresce in modo incontrollabile, producendo il messaggio di errore: "Spazio stack esaurito". Se è necessario uscire in anticipo da un ciclo, è meglio utilizzare l'istruzione EXIT DO.

Esempio

Il programma di seguito riportato utilizza l'istruzione DO...LOOP per creare un gioco. Viene visualizzato un carattere rappresentante uno sciatore, che si sposta lungo lo schermo tra le due righe di un tracciato; se lo sciatore tocca uno dei bordi, il programma termina. L'utente può evitare che ciò avvenga, spostando lo sciatore a destra e a sinistra mediante i tasti di direzione.

```
CLS : RANDOMIZE TIMER
ySciatore% = 21: xSciatore% = 55                'coordinate dello sciatore
yPista% = 24: xPista% = 51                      'coordinate della pista
desDirezione$ = CHR$(0) + CHR$(77): sinDirezione$ = CHR$(0) + CHR$(75)
pista$ = CHR$(176) + "          " + CHR$(176)

DO
    LOCATE 1, 1: PRINT "Punteggio ="; punteggio&        'stampa il punteggio
    punteggio& = punteggio& + 1

    LOCATE yPista%, xPista%: PRINT pista$           'stampa la pista
    cambio% = INT((RND * 3) - 1)
    xPista% = xPista% + cambio%                  'cambia la posizione della pista

    r$ = INKEY$
    IF r$ = sinDirezione$ THEN xSciatore% = xSciatore% - 1  'muove lo sciatore
                                                            'a sinistra
    IF r$ = desDirezione$ THEN xSciatore% = xSciatore% + 1  'muove lo sciatore
                                                            'a destra

    LOCATE ySciatore%, xSciatore%: PRINT CHR$(2);     'stampa lo sciatore
LOOP WHILE SCREEN(ySciatore% + 1, xSciatore%) <> 176  'esce dal ciclo se lo
                                                      'sciatore raggiunge la
                                                      'pista

END
```

EXIT

Si consulti anche

DEF FN, DO...LOOP, FOR...NEXT, FUNCTION, SUB

Scopo

EXIT consente di trasferire il controllo all'esterno di un ciclo o di una procedura prima che la condizione d'uscita venga soddisfatta o prima che la fine della procedura venga raggiunta. Questa istruzione deve essere utilizzata se si desidera che il programma risponda a un evento separato che potrebbe verificarsi prima del termine dell'esecuzione del ciclo.

Sintassi

```
EXIT {DEF | DO | FOR |FUNCTION |SUB}
```

L'istruzione EXIT deve essere utilizzata con una delle cinque parole chiave di QBasic riportate nella sintassi. È indispensabile includere una delle parole chiave comprese tra le parentesi graffe e la scelta di quale utilizzare dipende dal tipo di ciclo o di procedura in cui viene sfruttata l'istruzione EXIT.

Utilizzo

```
DO WHILE x% <100
   PRINT x%, x% ^ 2
   IF INKEY$ = "U" THEN EXIT DO
   x% = x% + 1
LOOP
```

EXIT DO è utilizzata per uscire da un ciclo, se l'utente immette la lettera U.

Descrizione

Le istruzioni EXIT DO ed EXIT FOR possono essere utilizzate per uscire da un ciclo DO...LOOP o FOR...NEXT. Quando un programma esegue una di queste due istruzioni, il controllo viene trasferito alla prima istruzione presente dopo il ciclo.

Se un programma contiene istruzioni FOR...NEXT o DO...LOOP nidificate, EXIT permette di uscire dal ciclo in esecuzione. Le istruzioni presenti in qualsiasi ciclo esteso a quello in cui si trova l'istruzione EXIT continuano a essere eseguite fino a quando il ciclo termina oppure viene individuata un'altra istruzione EXIT.

Le istruzioni EXIT DEF, EXIT FUNCTION ed EXIT SUB possono essere utilizzate per uscire da una procedura definita da un'istruzione DEF FN, FUNCTION o SUB. Quando un programma utilizza una di queste istruzioni, il controllo ritorna all'istruzione che segue la chiamata alla procedura. Se un programma utilizza EXIT DEF oppure EXIT FUNCTION prima che sia stata effettuata un'assegnazione alla funzione, quest'ultima restituisce un valore indeterminato.

Esempio

Il programma QBasic di seguito riportato richiama una funzione che verifica una stringa immessa dall'utente per controllare se costituisce un nome di file valido. La funzione verifica i caratteri presenti nella stringa e la lunghezza della stessa; la stringa può essere un nome di file con o senza estensione. Se il test non ha successo, la funzione utilizza l'istruzione EXIT FUNCTION per uscire prima del termine dell'esecuzione, impostando in questo modo un flag di validità a falso.

```
DECLARE FUNCTION NomeValido% (fileSpec$)

CONST FALSO = 0, VERO = NOT FALSO

CLS
PRINT "Inserisci un nomefile per verificarlo: ";
LINE INPUT f$
PRINT : PRINT "Questo è un";

IF NomeValido%(f$) THEN
    PRINT " nomefile valido."
ELSE
    PRINT " nomefile non valido."
END IF
END

'controlla il nomefile per verificare la lunghezza e la presenza di caratteri
'non validi
FUNCTION NomeValido% (fileSpec$) STATIC
    lunghezza% = LEN(fileSpec$)
    invalido$ = ""
    NomeValido% = FALSE
```

```
FOR i% = 0 TO 32         'crea una stringa di tutti i caratteri non validi
    invalido$ = invalido$ + CHR$(i%)
NEXT
invalido$ = invalido$ + "/\[]:|<>+=;," + CHR$(34)
FOR i% = 1 TO lunghezza%   'controlla l'occorrenza di ogni carattere
                           'invalido
    IF INSTR(invalido$, MID$(fileSpec$, i%, 1)) > 0 THEN
        EXIT FOR
    END IF
NEXT i%

IF i% < lunghezza% + 1 THEN EXIT FUNCTION    'la stringa ha un carattere
                                             'non valido
NomeValido% = VERO            'la stringa non ha caratteri non validi

punto% = INSTR(fileSpec$, ".") 'controllo dell'esistenza di un'estensione
IF punto% = 0 THEN            'controllo della lunghezza della stringa
    IF lunghezza% > 8 THEN
        NomeValido% = FALSO
    END IF
ELSEIF punto% > 9 OR punto% < (lunghezza% - 3) THEN 'controlla la
                                                    'lunghezza
                                                    'dell'estensione
    NomeValido% = FALSO
END IF
```

FOR...NEXT

Si consulti anche

DO...LOOP, EXIT, WHILE...WEND

Scopo

Un ciclo FOR...NEXT esegue una data sequenza di istruzioni per un determinato numero di volte. Deve essere utilizzato quando è necessario ripetere un gruppo di istruzioni per un numero di volte fissato, ad esempio per stampare i numeri da 1 a 100.

Sintassi

```
FOR contatore = inizio TO fine [STEP incremento]
   [istruzioni]
NEXT [contatore [,contatore...]]
```

contatore è una variabile numerica che conta il numero di volte in cui viene eseguito il ciclo. Esso può essere di tipo intero, intero lungo, a precisione semplice, a precisione doppia oppure un elemento di un array.

inizio è il valore assegnato al contatore quando inizia il ciclo e può essere una costante, una variabile oppure un'espressione.

fine è il valore per cui il ciclo verifica il **contatore** all'inizio di ogni passaggio. Se il valore del contatore è maggiore di quello di **fine** (oppure inferiore rispetto a **fine**, se **incremento** è un valore negativo), l'esecuzione del ciclo viene interrotta e il controllo viene trasferito all'istruzione successiva alla parola chiave **NEXT**. **fine** può essere una constante, una variabile o un'espressione.

incremento è la quantità in base alla quale viene incrementata la variabile del contatore (oppure diminuita, se **incremento** è un valore negativo) dopo ogni ripetizione del ciclo; può essere una costante, una variabile oppure un'espressione; nel caso venisse omesso, QBasic utilizza il valore 1.

Quando i cicli FOR...NEXT sono nidificati uno all'interno dell'altro, è possibile combinare l'istruzione NEXT di ogni ciclo in una singola istruzione NEXT, inserendo il nome di ogni contatore dopo la parola chiave NEXT.

Utilizzo

```
FOR terzo! = 1 TO 10 STEP 1 / 3
   PRINT terzo!;
NEXT terzo!
```

Questo ciclo conta da 1 a 10 in passi di un terzo e termina dopo 28 iterazioni.

```
FOR contatore% = 20 TO 1 STEP -1
   PRINT TAB(5); contatore%
NEXT contatore%
PRINT TAB(5); contatore%
PRINT "ACCIDENTI!"
```

Questo ciclo viene eseguito 20 volte; l'istruzione STEP diminuisce di 1 il valore della variabile contatore a ogni iterazione (si noti che, quando il ciclo termina, il contatore contiene il valore 0, non 1).

Descrizione

L'istruzione FOR...NEXT ripete l'esecuzione di un gruppo di istruzioni per un determinato numero di volte. QBasic, quando incontra un ciclo FOR...NEXT, memorizza innanzitutto il valore finale specificato e quindi imposta la variabile del contatore sul valore iniziale specificato; verifica inoltre l'istruzione STEP per controllare se il contatore di ciclo deve essere aumentato o diminuito nelle successive ripetizioni. Se non viene fornita alcuna istruzione STEP, QBasic incrementa la variabile del contatore di 1. Se, in questa fase preliminare, il valore del contatore è superiore rispetto al valore finale e l'incremento è positivo, QBasic non esegue il ciclo; in questo caso il controllo viene trasferito all'istruzione successiva a NEXT. In modo analogo, se l'incremento è negativo e il valore iniziale del contatore è inferiore al valore finale, il ciclo non viene eseguito.

Se il valore del contatore è compreso nell'intervallo specificato dai valori iniziali e finali, QBasic esegue le righe del programma successive all'istruzione FOR fino a quando non incontra l'istruzione NEXT, modificando quindi la variabile contatore in base al valore fornito nell'istruzione STEP. Se l'incremento è positivo il valore viene aggiunto alla variabile contatore, se invece è negativo, viene sottratto.

Si noti che il test effettivo viene sempre eseguito dopo la parola chiave FOR, all'inizio del ciclo. L'istruzione NEXT si limita a modificare in modo corretto il valore del contatore prima di restituire il controllo all'istruzione FOR. Questo significa che, al termine del ciclo, il valore del contatore è sempre maggiore (se l'incremento è positivo) oppure inferiore (se l'incremento è negativo) rispetto al valore finale specificato. Se, quando il controllo ritorna all'istruzione FOR, il valore del contatore è maggiore del valore finale (oppure inferiore nel caso di incremento negativo), il controllo viene trasferito all'istruzione che segue il ciclo. In caso contrario, il ciclo viene eseguito nuovamente.

I cicli FOR...NEXT possono essere nidificati. Ad esempio, nidificando un ciclo che esegue un conteggio da 1 a 10 in un ciclo che ne effettua uno da 1 a 5, il programma

conta da 1 a 10 per cinque volte. Inserendo in un'istruzione **NEXT** il nome del contatore di ogni ciclo in un ciclo nidificato **FOR...NEXT**, è possibile unire le istruzioni **NEXT** dei cicli. I due segmenti di codice di seguito riportati sono sintatticamente equivalenti.

```
FOR riga% = 1 TO 24
   LOCATE riga%, 1
   FOR strisce = 1 TO 20
      FOR colori% = 9 TO 12
            COLOR colori%
            PRINT "*";
      NEXT colori%
   NEXT strisce%
NEXT riga%
```

```
FOR riga% = 1 TO 24
   LOCATE riga%, 1
   FOR strisce = 1 TO 20
      FOR colori% = 9 TO 12
            COLOR colori%
            PRINT "*";
      NEXT colori%, strisce%, riga%
```

Errori

Ogni istruzione **FOR** deve essere seguita da un'istruzione **NEXT** corrispondente, proprio come ogni istruzione **NEXT** deve essere preceduta da **FOR**. Se l'istruzione **NEXT** viene omessa, QBasic visualizza il messaggio di errore: "FOR senza NEXT"; se invece QBasic individua un'istruzione **NEXT** che non è preceduta da **FOR**, visualizza il messaggio di errore: "NEXT senza FOR". Quest'ultimo messaggio viene visualizzato anche quando i cicli vengono inavvertitamente incrociati. La parte di codice di seguito riportata illustra un caso comune che produce questo tipo di errore.

```
FOR i% = 1 TO 20
   PRINT i%
   FOR j% 1 TO 29
      PRINT j%
   NEXT i%
   PRINT
NEXT j%
```

(NEXT i% / PRINT): Queste due righe sono state scambiate

Suggerimenti

Si noti che QBasic imposta il valore finale del ciclo prima di inizializzare il contatore al valore iniziale. Ad esempio, la parte di programma di seguito riportata stampa i numeri da 1 a 10 (non da 1 a 6, come si potrebbe ritenere).

```
i% = 5
FOR i% = 1 TO i% + 5
   PRINT i%;
NEXT
```

I cicli nidificati devono essere chiusi in ordine contrario rispetto a quello di apertura. Il ciclo **FOR...NEXT** non può essere chiuso fino a quando non sono stati chiusi tutti i cicli nidificati al suo interno.

Se in un'istruzione **NEXT** non viene specificatamente denominata una variabile, QBasic utilizza il contatore indicato nell'istruzione **FOR** inserita per ultima.

Esempio

I cicli FOR...NEXT risultano utili per operazioni che devono ripetere una serie di azioni per un numero esatto di volte. Il programma di seguito riportato stampa testi ampi per realizzazione di titoli di schermate o striscioni. Il testo visualizzato è codificato in istruzioni DATA che vengono lette all'interno del programma da una serie di cicli FOR...NEXT.

```
END FUNCTIONCONST FALSO = 0, VERO = NOT FALSO
DEF SEG = &H0: IF PEEK(&H49) = 7 THEN mono% = VERO  'controllo per video
                                                    'monocromatico

CLS : DEF SEG : RESTORE Testo

READ riga%, colonna%, tinta%, numColonne%
IF NOT mono% THEN COLOR tinta%

'stampa ciascuna lettera dello striscione una colonna alla volta
FOR nuovaColonna% = 1 TO numColonne%
    READ articolo$
    IF articolo$ = "F" THEN articolo$ = "12345678" 'se "F," riempie la colonna
    FOR i% = 1 TO LEN(articolo$)
        IF MID$(articolo$, i%, 1) = "*" THEN EXIT FOR    'se "*," inizia la
                                                         'lettera successiva
        spazio% = VAL(MID$(articolo$, i%, 1)) 'indice dello spazio da riempire
        LOCATE riga% + spazio%, colonna%
        PRINT CHR$(219);                               'riempie lo spazio
    NEXT i%
    IF i% = 1 THEN                  'cambia il colore della lettera successiva
        tinta% = tinta% + 1
        IF tinta% = 7 THEN tinta% = 9
        IF tinta% > 14 THEN tinta% = 1
        IF NOT mono% THEN COLOR tinta%
    END IF
    colonna% = colonna% + 1
NEXT nuovaColonna%
COLOR 14, 0: LOCATE 15, 28: PRINT "G U I D A  DI"
COLOR 11, 0: LOCATE 16, 31: PRINT "R I F E R I M E N T O"
END

'Ripartizione dei dati
Testo:
DATA      3,18,9,42
DATA      F,F,1238,168,1678,18,F,F,*                     '"Q"
DATA      F,F,1248,148,148,148,F,1235678,*               '"B"
DATA      F,F,124,14,14,F,F,*                            '"A"
DATA      123478,123478,148,148,1245678,1245678,*        '"S"
DATA      F,F,*                                          '"I"
DATA      F,F,18,18,1278,1278                            '"C"
```

GOTO

Scopo

L'istruzione GOTO può essere utilizzata per passare a una parte determinata del programma. In QBasic deve essere utilizzata raramente, dato che uno dei suoi scopi principali è quello di fornire la compatibilità con i programmi scritti con le versioni precedenti di Basic.

Sintassi

```
GOTO {etichettariga | numeroriga}
```

L'esecuzione passa all'istruzione in corrispondenza dell'etichettariga o del numeroriga e prosegue da quel punto fino a quando non viene individuata un'altra istruzione GOTO, oppure fino al termine del programma.

Utilizzo

```
GOTO 1000
```

Salta all'istruzione in corrispondenza della riga 1000 del modulo.

```
GOTO Stampa
```

Salta alla riga che presenta l'etichetta Stampa e prosegue l'esecuzione da quel punto.

Descrizione

GOTO trasferisce il controllo alla riga specificata; se essa contiene un commento o un'istruzione DATA, il controllo passa alla successiva istruzione che può essere eseguita. L'istruzione GOTO non può essere utilizzata per passare all'interno o all'esterno di un sottoprogramma o di una funzione definita dall'utente, mentre può essere sfruttata per dirigere il flusso di controllo all'interno di queste procedure.

Le istruzioni IF...THEN (illustrate nel Capitolo 3) possono comprendere un'istruzione GOTO implicita dopo la clausola THEN, come riportato nel seguente esempio.

```
IF y% = 50 THEN 1000
```

In questo caso, l'esecuzione passa alla riga 1000 quando y% vale 50. Questa istruzione GOTO implicita funziona tuttavia soltanto con i numeri di riga; se la riga destinazione avesse un'etichetta, sarebbe indispensabile inserire la parola chiave GOTO.

```
IF x% >= 100 THEN GOTO Overflow
```

Esempio

Le strutture di controllo di QBasic, quali le istruzioni IF poste su più righe e i cicli DO, hanno diminuito la necessità di utilizzare le istruzioni GOTO. L'esempio di seguito riportato si avvale dell'istruzione GOTO per verificare l'immissione del nome corretto da parte dell'utente; il sottoprogramma che lo segue utilizza la programmazione strutturata per eseguire lo stesso compito: questo tipo di programmazione consente di creare un codice compatto ed efficace, più semplice da gestire.

```
10  lc$ = "abcdefghijklmnopqrstuvwxyz"
20  uc$ = UCASE$(lc$)
30  nu$ = "0123456789-+/."
40  valido$ = uc$ + lc$      'una stringa contenente tutti i caratteri validi
50  buffer$ = ""             'buffer per il nome
60  max% = 30                'numero massimo di caratteri

70  CLS : LOCATE 10, 1, 1: PRINT "Inserisci il tuo nome > ";
```

```
80  lunghezza% = 0
90  tastoPremuto$ = INKEY$
100 IF tastoPremuto$ = "" THEN 90          'ripete fino a quando l'utente
                                           'inserisce un carattere
110 IF tastoPremuto$ = CHR$(13) THEN 230   'termina quando l'utente preme Invio

120 IF tastoPremuto$ <> CHR$(8) THEN 180   'salta se l'utente non preme
                                           'Backspace
130 IF lunghezza% = 0 THEN 90
140 PRINT CHR$(29); " "; CHR$(29);
150 lunghezza% = lunghezza - 1
160 buffer$ = LEFT$(buffer$, lunghezza%)
170 GOTO 90

180 IF lunghezza% = max% OR INSTR(valido$, tastoPremuto$) = 0 THEN 90
190 buffer$ = buffer$ + tastoPremuto$
200 PRINT tastoPremuto$;
210 lunghezza% = lunghezza% + 1
220 GOTO 90

230 LOCATE 20, 1: PRINT buffer$
240 END

'questo programma equivale al precedente ma utilizza tecniche di
'programmazione strutturata

SUB OttieneInput (max%, buffer$) STATIC
    SHARED valido$
    lunghezza% = 0
    DO
       DO
           tastoPremuto$ = INKEY$
       LOOP WHILE tastoPremuto$ = ""       'ripete fino a quando l'utente
                                           'inserisce un carattere
       IF tastoPremuto$ = CHR$(8) THEN     'controlla se l'utente ha premuto
                                           'Backspace
          IF lunghezza% > 0 THEN
              lunghezza% = lunghezza% - 1
              PRINT CHR$(29); " "; CHR$(29);
              buffer$ = LEFT$(buffer$, lunghezza%)
          END IF
       ELSE
          IF lunghezza% < max% AND INSTR(valido$, tastoPremuto$) THEN
              buffer$ = buffer$ + tastoPremuto$
              PRINT tastoPremuto$; : lunghezza% = lunghezza% + 1
          END IF
       END IF
    LOOP UNTIL tastoPremuto$ = CHR$(13)  'termina quando l'utente preme Invio
END SUB
```

ON...GOSUB

Si consulti anche

GOSUB...RETURN, ON...GOTO, SELECT CASE

Scopo

L'istruzione ON...GOSUB passa il controllo a una subroutine QBasic; il valore di un'espressione determina quale. ON...GOSUB è analoga all'istruzione ON...GOTO, tranne per il fatto che il controllo alla fine ritorna alla parte del programma da cui è partito.

Sintassi

```
On espressione GOSUB {riga 1|etichetta1} [,{riga2|etichetta2}...]
```

espressione è una variabile numerica o un'espressione; QBasic la converte in un intero e quindi applica il valore all'elenco dei numeri di riga o alle etichette. L'elenco viene utilizzato come una *tabella di salto* che indica la riga a cui viene trasferito il controllo. Se il valore di espressione è 1, il programma salta alla riga indicata da riga1 o etichetta1; se il valore dell'espressione è 2, il programma salta alla riga indicata da riga2 o etichetta2 e così via. Si noti che questo elenco può essere composto da un insieme di etichette e di numeri di riga.

Utilizzo

```
ON INT(RND * 2) + 1 GOSUB Inizio, Fine
```

Passa a una delle due subroutine, in base al valore del numero casuale.

Descrizione

ON...GOSUB rappresenta un'alternativa rispetto all'istruzione ON...GOTO. In modo analogo a quest'ultima, essa consente al programma di passare a uno dei diversi numeri di riga o etichette specificati, in base al valore dell'espressione; l'esecuzione prosegue da quel punto. Al contrario di ON...GOTO, l'indirizzo di destinazione è l'inizio di una subroutine QBasic. Quando il programma individua l'istruzione RETURN che contrassegna la fine di una subroutine, il controllo ritorna all'istruzione successiva a ON...GOSUB.

Se il valore dell'espressione è 0, oppure un numero superiore rispetto al numero di voci presenti nella tabella di salto, il controllo passa all'istruzione che segue ON...GOSUB e non viene eseguita nessuna subroutine.

Non è necessario utilizzare un numero di riga diverso o un'etichetta differente per ogni voce della tabella di salto; lo stesso numero di riga o di etichetta può essere utile in diverse voci. Nell'esempio di seguito riportato, il programma passa al numero di riga 200 quando la variabile dove% contiene il valore 2, 4 o 7.

```
ON dove% GOSUB Inizio, 200, 300, 200, File, Stampante, 200, Fine
```

Commenti

Sebbene QBasic possa accettare numeri interi compresi nell'intervallo da 0 a 255 come valori dell'espressione in un'istruzione ON...GOSUB, non è possibile elencare un numero superiore a 60 righe o etichette nella tabella di salto. In pratica si è ancora più limitati, dato che è necessario che tutta l'istruzione, incluso l'elenco dei numeri di riga e delle etichette, sia compresa in una sola riga di programma.

Errori

Se l'espressione utilizzata nell'argomento ON ha un valore inferiore a 0 o superiore a 255, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita".

Esempio

Dato che ON...GOSUB è in grado di passare a subroutine diverse in relazione a differenti valori di un'espressione, questa istruzione risulta utile quando si tratta di programmare test costituiti da domande che prevedono diverse risposte e menu. Il programma di seguito riportato visualizza una domanda che prevede diverse risposte e quindi utilizza un'istruzione ON...GOSUB per eseguire la routine che corrisponde alla risposta ricevuta.

```
COLOR 0, 2: CLS : COLOR 14, 4: PRINT SPACE$(240)
LOCATE 2, 30: PRINT "****SCELTA MULTIPLA***";
LOCATE 4, 1: PRINT STRING$(80, "_"); : COLOR 0, 2
LOCATE 10, 25: PRINT "Ti stai divertendo con questo programma?";

DO
    PLAY "L62 04 T250 D G G D G# A A 03 D G G D A A# A# 02 D G G D A# B B"
    LOCATE 13, 29, 1: PRINT "<S>i,  <N>o,  <F>orse";
    LOCATE 15, 40
    r$ = UCASE$(INPUT$(1))                     'ottiene input dall'utente
    ON INSTR("SNF", r$) GOSUB Si, No, Forse    'salta a Si, No o Forse
LOOP WHILE risposta$ = ""
LOCATE 18, 40 - (LEN(risposta$) \ 2)
PRINT risposta$;
PLAY tono$
END

Forse:
    tono$ = "T255 L15 01 A D C E F A 02 D E F A E C"
    risposta$ = "Il tuo suono è indeciso!"
    RETURN
Si:
    tono$ = "L8 02 T240 D.. E. P16 ML A G B."
    risposta$ = "Il tuo suono è assai positivo!"
    RETURN
No:
    tono$ = "T200 0314 G C D E L2 F L4 E C L2 D P64 D L4 E L1C"
    risposta$ = "Il tuo suono è veramente negativo!"
    RETURN
```

ON...GOTO

Si consulti anche

GOTO, ON...GOSUB, SELECT CASE

Scopo

L'istruzione ON...GOTO passa il controllo a una data riga del programma. Spesso è necessario che un programma esegua un compito diverso a seconda di un evento che si verifica durante l'esecuzione, e che potrebbe essere il risultato di un test o di un'immissione di dati da parte dell'utente. L'istruzione ON...GOTO rappresenta il metodo tradizionale per sottoporre a test un evento e trasferire il controllo all'indirizzo corretto.

Sintassi

```
On espressione GOTO {riga1|etichetta1} [,{riga2|etichetta2}...]
```

espressione è una variabile numerica o un'espressione; QBasic la converte in un intero e quindi applica il valore all'elenco dei numeri di riga o alle etichette. L'elenco viene utilizzato come una tabella di salto che indica la riga a cui viene trasferito il controllo. Se il valore di espressione è 1, il programma salta alla riga indicata da riga1 o etichetta1; se il valore dell'espressione è 2, il programma salta alla riga indicata da riga2 o etichetta2 e così via. Si noti che questo elenco può essere composto da un insieme di etichette e di numeri di riga.

Utilizzo

```
ON INT(RND * 3) + 1 GOTO Uno, Due, Tre
```

Passa in modo casuale alla routine di destinazione che inizia in corrispondenza dell'etichetta Uno, Due o Tre.

Descrizione

L'utilizzo dell'istruzione ON...GOTO rappresenta il metodo tradizionale per creare le tabelle di salto in Basic. Il valore dell'espressione fornito è un indice dell'elenco di numeri di riga o di etichette che segue la parola chiave GOTO. Quando l'istruzione viene eseguita, QBasic valuta l'espressione e quindi trasferisce il controllo alla riga specificata dalla voce di tabella corrispondente. Se il valore dell'espressione è 0, oppure un numero superiore rispetto al numero di voci presenti nella tabella di salto, il controllo passa direttamente all'istruzione che segue ON...GOTO.

Non è necessario utilizzare un numero di riga diverso o un'etichetta differente per ogni voce della tabella di salto; lo stesso numero di riga o di etichetta può essere utile in diverse voci. Nell'esempio di seguito riportato, il programma passa al numero di riga 200 quando la variabile dove% contiene il valore 2, 4 o 7.

```
ON dove% GOTO Inizio, 200, 300, 200, File, Stampante, 200, Fine
```

L'istruzione ON...GOTO non può essere utilizzata per trasferire il controllo all'interno o all'esterno di un sottoprogramma o di una funzione definita dall'utente, mentre può essere sfruttata per dirigere il flusso di controllo all'interno di queste procedure.

QBasic fornisce un metodo più flessibile per realizzare le tabelle di salto, l'istruzione SELECT CASE. Uno dei vantaggi di questa istruzione è rappresentato dal fatto che consente di specificare un intervallo di valori in base ai quali il programma passa il controllo a un singolo indirizzo.

Commenti

Sebbene QBasic possa accettare numeri interi compresi nell'intervallo da 0 a 255 come valori dell'espressione in un'istruzione ON...GOTO, non è possibile elencare un numero superiore a 60 righe o etichette nella tabella di salto. In pratica si è ancora più limitati, dato che è necessario che tutta l'istruzione, incluso l'elenco dei numeri di riga e delle etichette, sia compresa in una sola riga di programma.

Errori

Se l'espressione utilizzata nell'argomento ON ha un valore inferiore a 0 o superiore a 255, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita".

Esempio

Nei giochi e nei programmi per l'autoaddestramento, è necessario essere in grado di ricompensare il giocatore che ottiene un successo. Il programma di seguito riportato utilizza l'istruzione ON...GOTO per indicare al computer di suonare un pezzo musicale con una durata che corrisponde alle dimensioni del punteggio ottenuto in base a una chiamata alla funzione RND.

```
RANDOMIZE TIMER:   'forza un nuovo numero casuale ogni volta

CLS : LOCATE 10, 30: PRINT "IL TUO PUNTEGGIO ...";
punteggio% = RND * 5000
PRINT punteggio%

ON (punteggio% \ 1000) + 1 GOTO Brevissimo, Breve, Lungo, Lunghissimo

Brevissimo:
    PLAY "L4 T200 ML 02 G E G F D 01 B 02 C.. MN"
    SYSTEM
Breve:
    PLAY "P32 L4 T200 ML 01 G 02 C D E2 C E 02 D C D E F"
    SYSTEM
Lungo:
    PLAY "L4 T200 ML 01 G 02 C D E.. P16 MS E ML E D C D. D"
    SYSTEM
Lunghissimo:
    PLAY "L4 T200 ML 01 G 02 C D E.. P16 MS E ML E D C D C 01 A P48"
    SYSTEM
END
```

WHILE...WEND

Si consulti anche

DO...LOOP, FOR...NEXT

Scopo

L'istruzione WHILE...WEND consente di ripetere una serie di istruzioni soltanto finché rimane vera una determinata condizione (ad esempio, fino a quando non viene raggiunta la fine del file).

Sintassi

```
WHILE condizione
   [istruzioni]
WEND
```

condizione è una qualsiasi variabile numerica o espressione che rappresenta il valore vero (un valore diverso da zero) o falso (zero). QBasic esegue le istruzioni nel ciclo tra le parole chiave WHILE e WEND, purché la **condizione** sia vera; quando diventa falsa, il controllo passa all'istruzione che segue WEND e l'esecuzione prosegue da questo punto.

Utilizzo

```
WHILE INKEY$ = ""
WEND
```

Esegue un ciclo continuo fino a quando un utente preme un tasto.

```
WHILE NOT EOF(1)
   LINE INPUT #1 buffer$
   PRINT buffer$
WEND
```

Legge righe successive da un file sequenziale nella variabile stringa buffer$ e le visualizza sullo schermo. Questa azione prosegue fino a quando non si raggiunge la fine del file.

Descrizione

WHILE...WEND viene definito un ciclo con test iniziale, dato che il test logico viene eseguito all'inizio. Se l'espressione è già falsa quando viene individuata l'istruzione WHILE, QBasic ignora il ciclo e non esegue le istruzioni al suo interno.

Il corpo del ciclo WHILE...WEND può contenere qualsiasi numero di istruzioni, inclusi altri cicli WHILE...WEND, che possono essere nidificati a qualsiasi livello. QBasic dispone dell'istruzione DO...LOOP, che fornisce una flessibilità maggiore rispetto all'istruzione WHILE...WEND.

Errori

Ogni istruzione WHILE deve essere seguita da una WEND corrispondente, proprio come ogni istruzione WEND deve essere preceduta da WHILE. Se il programma esegue un'istruzione WEND senza aver individuato in precedenza WHILE, QBasic visualizza il messaggio di errore: "WEND senza WHILE". A un'istruzione WHILE priva di WEND corrisponde il messaggio di errore: "WHILE senza WEND".

Esempio

Dato che un ciclo WHILE...WEND viene eseguito fino a quando una determinata condizione rimane vera, esso risulta particolarmente adatto nel caso di procedure che devono controllare eventi esterni, ad esempio l'immissione di dati da parte dell'utente. Il programma di seguito riportato utilizza le istruzioni WHILE...WEND per passare tra due visualizzazioni grafiche; una delle due passa all'altra, in base alla pressione di un tasto.

```
'questo programma richiede almeno una scheda grafica CGA
SCREEN 1: LINE (0, 0)-(320, 200), 1, BF
VIEW (32, 4)-(288, 196), 0, 15
CLS 1
esc$ = CHR$(27)
WHILE tastoPremuto$ <> esc$
    LINE (78, 75)-(178, 75), 3                'disegna la linea superiore
    LINE (78, 125)-(178, 125), 3              'disegna la linea inferiore
    PSET (78, 75), 3: DRAW "C3 H8 BD16 E8"    'disegna le freccie
    PSET (178, 75), 3: DRAW "C3 E8 BD16 H8"
    PSET (86, 125), 3: DRAW "C3 BU8 G8 F8"
    PSET (170, 125), 3: DRAW "C3 BU8 F8 G8"
    tastoPremuto$ = ""
    WHILE tastoPremuto$ = ""
        tastoPremuto$ = INKEY$
    WEND
    IF tastoPremuto$ <> esc$ THEN
        PSET (78, 75), 3: DRAW "C0 H8 BD16 E8"     'cancella le freccie
        PSET (178, 75), 3: DRAW "C0 E8 BD16 H8"
```

```
        PSET (86, 125), 3: DRAW "C0 BU8 G8 F8"
        PSET (170, 125), 3: DRAW "C0 BU8 F8 G8"
        tastoPremuto$ = ""
        WHILE tastoPremuto$ = ""
            tastoPremuto$ = INKEY$
        WEND
    END IF
WEND
END
```

Capitolo 3

Decisioni e operatori

INTRODUZIONE

Una persona normale prende in media centinaia di decisioni ogni giorno; nuove situazioni e nuovi stimoli costringono ognuno a reagire e a modificare i propri comportamenti. Anche i programmi devono confrontare le nuove informazioni e modificare in base a ciò le proprie prestazioni. L'utilità dei computer sarebbe veramente minima se le loro risposte fossero sempre fisse e inalterabili.

Le strutture decisionali sono vitali per la programmazione. Un'azione semplice (e tuttavia fondamentale) come quella di accettare la scelta di un'opzione da parte di un utente richiede una decisione: ad esempio, se l'utente immette una "A" (per Aggiungere), il programma aggiunge nuove informazioni, mentre se immette "U" (per Uscire) il programma termina. Le istruzioni, le funzioni e gli operatori delle categorie decisionali di QBasic consentono di creare programmi in grado di prendere queste importanti decisioni.

In questo libro vengono riepilogate le capacità decisionali di QBasic e vengono inoltre offerti alcuni suggerimenti relativi al loro utilizzo. Nella Tabella 3.1 vengono elencate le istruzioni, le funzioni e gli operatori decisionali di QBasic.

IF - L'ISTRUZIONE DECISIONALE DI CARATTERE GENERALE

QBasic mutua dall'inglese l'istruzione IF, che viene utilizzata per prendere decisioni di carattere generale. Si consideri ad esempio la seguente frase:

Se il sole splende, andrò a un picnic, in caso contrario rimarrò a casa.

Questa frase può essere riscritta comprendendo le parole "allora" e "altrimenti".

Se il sole splende, allora andrò a un picnic, altrimenti rimarrò a casa.

Tabella 3.1 Le istruzioni, le funzioni e gli operatori decisionali di QBasic

Istruzione, funzione od operatore	Descrizione
AND	Operatore logico di congiunzione
+	Operatore di concatenamento stringhe
EQV	Operatore logico di equivalenza
IF	Istruzione condizionale
IMP	Operatore logico di implicazione
MOD	Operatore aritmetico di modulo
NOT	Operatore logico di negazione
OR	Operatore logico di disgiunzione
SELECT CASE	Istruzione condizionale di confronto
XOR	Operatore logico "o esclusivo"

Ora è possibile tradurre questa frase nelle istruzioni di QBasic, ovvero:

```
IF soleSplende THEN
   GOTO Picnic
ELSE
   GOTO Casa
END IF
```

L'espressione che segue la parola chiave IF (in questo caso soleSplende) viene definita *antecedente.*

L'effetto dell'istruzione IF è evidenete: se l'antecedente è vero, allora le istruzioni successive alla parola chiave THEN vengono eseguite; se l'antecedente è falso, vengono invece eseguite le istruzioni successive alla parola chiave ELSE. Nell'esempio utilizzato, se l'espressione soleSplende è vera, il programma passa a Picnic; se invece è falsa, il programma passa a Casa.

Si osservi che per QBasic vero e falso sono soltanto dei numeri: falso è 0 e vero è qualsiasi numero diverso da 0. Nell'esempio precedente, soleSplende potrebbe essere una variabile rappresentata dal valore 0 quando il sole non splende e da qualsiasi altro numero quando invece splende. QBasic genera internamente il valore -1 per il vero, anche se qualsiasi numero diverso da 0 viene riconosciuto come vero.

OPERATORI RELAZIONALI

A questo punto vengono esaminati gli *operatori relazionali* di QBasic, ovvero =, <>, >, <, >= e <=, definiti nella Tabella 3.2.

Avendo ben chiaro in che modo QBasic riconosce il vero e il falso, è possibile affrontare il funzionamento degli operatori relazionali. È necessario tenere presente che QBasic restituisce il valore -1 se un'espressione è vera e 0 se l'espressione è falsa. Tutti gli operatori di seguito riportati confrontano un valore con un altro.

Tabella 3.2 Gli operatori relazionali

Operatore relazionale	Simbolo matematico	Esempio	Significato
=	=	x = y	Uguale a
<>	≠	x <> y	Diverso da
>	>	x > y	Maggiore di
<	<	x < y	Minore di
>=	≥	x >= y	Maggiore o uguale a
<=	≤	x <= Y	Minore o uguale a

Uguaglianza

L'operatore = controlla l'uguaglianza. Quando QBasic incontra l'espressione:

```
x = y
```

confronta x con y; se x è uguale a y l'espressione è vera, altrimenti è falsa. Ad esempio, l'espressione 150 = 200 è falsa.

Disuguaglianza

L'operatore <> controlla la disuguaglianza. Quando QBasic incontra l'espressione:

```
x <> y
```

confronta x con y; se x non è uguale a y l'espressione è vera, altrimenti è falsa. Ad esempio, l'espressione 150 <> 200 è vera.

Maggiore di

L'operatore > controlla la relazione tra i valori indicati. L'espressione:

```
x > y
```

è vera se x è maggiore di y, altrimenti è falsa.

Minore di

In modo analogo all'operatore >, anche l'operatore < controlla la relazione tra i valori indicati. L'espressione:

x < y

è vera se x è minore a y, altrimenti è falsa. Ad esempio, l'espressione 150 < 200 è vera.

Maggiore o uguale a

L'operatore >= controlla la relazione tra i valori indicati, comprendendo anche l'uguaglianza. L'espressione:

x >= y

confronta i valori di x e y; se x è maggiore o uguale a y l'espressione è vera, altrimenti risulta falsa. Ad esempio, l'espressione 150 >= 200 è falsa, mentre l'espressione 100 >= 100 è vera.

Minore o uguale a

In modo analogo all'operatore >=, anche l'operatore <= controlla la relazione tra i valori indicati, comprendendo l'uguaglianza. L'espressione:

x <= y

è vera se x è minore o uguale a y, altrimenti risulta falsa.

Esempi di utilizzo degli operatori relazionali

Si supponga che temp sia una variabile che contiene il valore della temperatura. Per essere certi di non andare a un picnic a meno che la temperatura non sia superiore ai 20 gradi, è necessario utilizzare l'istruzione IF di seguito riportata.

```
IF temp > 20 THEN
   GOTO Picnic
ELSE
   GOTO Casa
ENDIF
```

Se si desidera andare alle terme ogniqualvolta la temperatura dell'acqua in spiaggia è di 20 gradi o inferiore, è possibile prendere la propria decisione utilizzando la seguente istruzione IF.

```
IF TempAcqua <= 20 THEN
   GOTO Terme
ELSE
   GOTO Spiaggia
END IF
```

Si potrebbe anche decidere di diversificare le proprie attività in base al tempo atmosferico. Si supponga di voler andare al cinema in caso di pioggia, andare in spiaggia se il sole splende, costruire un pupazzo di neve se nevica e se invece nessuna

di queste attività fosse possibile, di schiacciare un pisolino. Tutto ciò può essere rappresentato in QBasic utilizzando la parola chiave ELSEIF in un'istruzione IF...THEN nel seguente modo.

```
IF tempo = sole THEN
   GOTO Spiaggia
ELSEIF tempo = pioggia THEN
   GOTO Cinema
ELSEIF tempo = neve THEN
   GOTO CostruirePupazzo
ELSE
   GOTO Pisolino
END IF
```

In questa istruzione, la variabile tempo rappresenta uno dei molti valori possibili relativi al tempo atmosferico. QBasic valuta ogni istruzione IF e ELSEIF, cercando di individuare la corrispondenza; questo esempio contiene soltanto tre istruzioni, d'altra parte è necessario tenere presente che, se si dovessero utilizzare un numero eccessivo di istruzioni ELSEIF, il programma potrebbe risultare difficile da leggere e da comprendere. Fortunatamente QBasic offre un'alternativa, l'istruzione SELECT CASE.

GLI OPERATORI LOGICI

Uno dei pregi di QBasic è la possibilità di combinare espressioni semplici per crearne di complesse, mediante gli operatori logici AND, EQV, IMP, NOT, OR e XOR.

Viene ora ampliato l'esempio precedente relativo al picnic, in modo da inserire ulteriori dettagli. Ad esempio, si potrebbe specificare che il sole deve splendere e che la temperatura deve essere superiore ai 20 gradi per decidere di andare al picnic.

```
IF soleSplende AND temp > 20
   GOTO Picnic
ELSE
   GOTO Casa
ENDIF
```

Si noti che l'operatore AND funziona in modo analogo alla stessa congiunzione in inglese, o alla congiunzione italiana "e". È necessario inserire un operando prima e dopo AND, come nella seguente espressione:

```
questo AND quello
```

L'operatore AND restituisce il vero soltanto quando entrambi gli operandi sono veri. Nell'esempio precedente, il sole deve splendere e la temperatura deve essere superiore ai 20 gradi per decidere di andare al picnic.

L'operatore OR invece restituisce il vero se uno dei due operandi è vero. Ad esempio, se la temperatura dell'acqua in spiaggia è inferiore a 20 gradi o il sole non splende, si potrebbe decidere di recarsi alle terme. L'istruzione IF di seguito riportata esprime quelle condizioni.

```
IF TempAcqua < 20 OR SoleNonSplende THEN
   GOTO Terme
```

```
ELSE
   GOTO Spiaggia
END IF
```

L'operatore NOT è diverso dagli altri operatori logici, in quanto è un *operatore unario*, ovvero richiede un solo operando. Questo operatore, come è implicito nel suo nome, inverte il valore dell'operando: se quest'ultimo è vero, il risultato di un'operazione NOT è falso; se l'operando è falso, il risultato di un'operazione NOT è vero. Quindi, l'espressione precedente può anche essere strutturata nel seguente modo:

```
IF TempAcqua < 20 OR NOT SoleNonSplende THEN
   GOTO Terme
ELSE
   GOTO Spiaggia
END IF
```

Gli operatori logici EQV, IMP e XOR consentono di creare espressioni complesse, dato che sono in grado di verificare contemporaneamente due espressioni. L'operatore EQV restituisce un'epressione vera se entrambi gli operandi sono veri oppure entrambi sono falsi. L'operatore IMP restituisce il falso soltanto se il primo operando è vero e il secondo è falso. Infine, l'operatore XOR restituisce il vero soltanto quando solo uno degli operandi è vero; se entrambi gli operandi sono veri (o falsi), XOR restituisce il falso.

SELECT CASE

L'istruzione SELECT CASE di QBasic è in sostanza equivalente all'istruzione IF, ma utilizza un formato più chiaro. Ad esempio, le istruzioni di seguito riportate utilizzano SELECT CASE per eseguire lo stesso compito dell'esempio precedente, che invece si avvaleva dell'istruzione IF.

```
SELECT CASE tempo
   CASE sole
      GOTO Spiaggia
   CASE pioggia
      GOTO Cinema
   CASE neve
      GOTO CostruirePupazzo
   CASE ELSE
      GOTO Pisolino
END SELECT
```

Si supponga di voler restare a casa anche quando soffia un forte vento. Per indicare questo, è necessario modificare una clausola ELSEIF nell'istruzione IF, nel seguente modo:

```
ELSEIF tempo = pioggia OR tempo = vento THEN
   GOTO Casa
```

Con l'istruzione SELECT CASE è invece sufficiente aggiungere vento nella clausola corretta:

```
CASE pioggia, vento
   GOTO Casa
```

L'OPERATORE MOD

L'operatore MOD restituisce il resto di un'operazione di divisione intera. Ad esempio, il risultato di 5 MOD 2 è 1, dato che 5 diviso per 2 dà come risultato 2 con il resto di 1.

I programmatori sfruttano spesso l'operatore MOD come operatore "divisibile per". Ad esempio, per determinare se un anno è bisestile è necessario sapere se è divisibile per 4 ma non per 100 (a meno che sia anche divisibile per 400); a questo scopo è possibile utilizzare l'operatore MOD, come illustrato nel seguente esempio.

```
CLS
DIM anno AS INTEGER, per4 AS INTEGER, per100 AS INTEGER, per400 AS INTEGER

INPUT "Immettere un anno con quattro cifre (ad esempio: 1989)"; anno
per4 = anno MOD 4  'divisibile per 4?
per100 = anno MOD 100    'divisibile per 100?
per400 = anno MOD 400    'divisibile per 400?

IF (per4 = 0 AND per100 <> 0) OR per400 = 0 THEN
   PRINT "Si,"; anno; "è un anno bisestile."
ELSE
   PRINT "No,"; anno; "non è un anno bisestile."
END IF
```

L'OPERATORE + (CONCATENAZIONE)

QBasic offre un supporto integrato per la combinazione (o *concatenazione*) di due stringhe; oltre a sommare due numeri, con l'operatore + è anche possibile concatenare due stringhe. La concatenazione viene spesso utilizzata per combinare gli elementi separati di un nome in una stringa singola. Ad esempio, dopo l'esecuzione delle istruzioni:

```
titolo$ = "Sig."
nome$ = "Carlo"
cognome$ = "Conti"

completo$ = titolo$ + " " + nome$ + " " + cognome$
```

il valore di completo$ è "Sig. Carlo Conti". Si noti che anche l'ultima istruzione concatena degli spazi alla stringa in modo da poter separare le diverse parti di cui è composto il nome.

Bibliografia

Bergmann, Merrie, et al. *The Logic Book.* New York, N.Y.: Random House, 1980.

Prata, Stephen, with Harry Henderson. *The Waite Group's Microsoft QuickBASIC Primer Plus.* Redmond, Wash.: Microsoft Press, 1990.

AND

Si consulti anche

EQV, IF, IMP, MOD, NOT, OR, XOR

Scopo

AND è un operatore sia logico che aritmetico. Come operatore logico, restituisce il vero soltanto se entrambi gli operandi sono veri. Come operatore aritmetico, esegue operazioni AND con i bit corrispondenti di due valori.

Sintassi

```
IF op1 AND op2 THEN...
```

oppure

```
risultato% = op1 AND op2
```

Gli operandi op1 e op2 nel primo caso possono essere espressioni logiche oppure valori di qualsiasi tipo numerico (intero, intero lungo, a precisione semplice o a precisione doppia); nel secondo caso possono essere di qualsiasi tipo numerico, ma il risultato è sempre un valore di tipo intero o intero lungo.

risultato% è la variabile che contiene il risultato di un'operazione aritmetica AND.

Utilizzo

```
IF ricco% AND famoso% THEN
   PRINT "Sei sposato?"
END IF
```

Se le variabili ricco% e famoso% sono entrambe vere (hanno un valore diverso da zero), il messaggio viene visualizzato.

Descrizione

L'operatore logico AND può essere utilizzato in un'espressione logica per determinare se entrambi gli operandi sono veri. Ad esempio l'espressione:

```
var1 > 2 AND var2 < 4
```

è vera soltanto quando le espressioni var1 > 2 e var2 < 4 sono entrambe vere.

L'operatore AND può anche essere utilizzato in modo aritmetico per eseguire un confronto bit per bit degli operandi. Se i bit corrispondenti negli operandi sono 1, il bit risultante è 1; in caso contrario (se uno o entrambi dei bit è 0), il risultato è 0. Si consulti la tabella di verità riportata nella Tabella 3.3 per ulteriori dettagli.

Nella Figura 3.1 è illustrato l'utilizzo di questo operatore.

L'operatore AND accetta operandi a precisione semplice e a precisione doppia, convertendoli però in interi o interi lunghi prima di elaborarli; il risultato è sempre un valore di tipo intero o intero lungo.

Tabella 3.3 La tabella di verità AND

p	q	p AND q
1	1	1
1	0	0
0	1	0
0	0	0

```
L'istruzione aritmetica AND

      262 AND 39

viene calcolata come segue:

      262 decimale = 100000110 binario
   AND 39 decimale = 000100111 binario
      ---------------------------------
        6 decimale = 000000110 binario
                              ||
      Entrambi i bit in queste colonne sono 1
```

Figura 3.1 AND aritmetico

Errori

Gli operandi di un'operazione AND possono essere di qualsiasi tipo numerico (intero, intero lungo, a precisione semplice, a precisione doppia), compresi nell'intervallo da -2147483648 a 2147483647; in caso contrario, QBasic visualizza il messaggio di errore: "Overflow".

Esempio

L'esempio di seguito riportato illustra l'utilizzo dell'operatore AND per determinare l'idoneità di una persona a ottenere un un prestito.

```
CLS
PRINT "Applicazione PRESTITI"
PRINT

INPUT "Ammontare del prestito desiderato"; ammontare#
INPUT "Qual'è lo stipendio annuale del richiedente"; stipendio#

PRINT
IF ammontare# > 10000000 EQV stipendio# > 20000000 THEN
    PRINT "poichè l'ammontare del prestito richiesto è"
    PRINT "appropriato per lo stipendio del richiedente,"
    PRINT "il prestito di L."; ammontare#; "è approvato."
ELSE
    PRINT "Il richiedente non soddisfa le nostre condizioni; ";
    PRINT "il prestito è negato."
END IF
```

+ (Concatenazione)

Si consulti anche

CONST, FRE

Scopo

L'operatore +, quando utilizzato con stringhe, concatena due o più stringhe in una sola; può essere ad esempio utilizzato per combinare un nome proprio e un cognome in una sola stringa.

Sintassi

```
risultato$ = stringa1$ + stringa2$ [ + stringa3$... ]
```

risultato$ è una variabile stringa che contiene la concatenazione di stringa1$, stringa2$, stringa3$ e così via.

Utilizzo

```
globale$ = prefisso$ + NomeBase$
```

Combina il contenuto di prefisso$ e NomeBase$ in globale$.

```
NomeCompleto$ = Nome$ + " " + Cognome$
```

Combina il contenuto di Nome$ e Cognome$ (separati da uno spazio) in NomeCompleto$.

Descrizione

L'operatore + esegue la concatenazione di stringhe, ovvero il processo di combinazione di diverse stringhe o espressioni stringa in una sola stringa più lunga. Ad esempio l'operazione:

```
"ABCDE" + "FGHIL"
```

dà come risultato la stringa "ABCDEFGHIL". L'operatore + consente di combinare diverse stringhe in una sola operazione, ad esempio:

```
"ABCDE" + FGHIL" + "MNOPQ" + "RSTUVZ"
```

dà come risultato la stringa "ABCDEFGHILMNOPQRSTUVZ".

Commenti

QBasic non consente di utilizzare la concatenazione di stringhe in un'istruzione CONST.

Quando si concatena una stringa a lunghezza variabile con un'altra stringa, QBasic crea una nuova stringa contenente le due originali. Questo si verifica persino se vengono semplicemente aggiunti alcuni caratteri alla stessa stringa; ad esempio, la concatenazione di stringhe di seguito riportata aggiunge alcuni caratteri a lungaStringa$.

```
lungaStringa$ = lungaStringa$ + "*-*-*-*-*"
```

QBasic crea una nuova stringa, copia il contenuto precedente di lungaStringa$ in essa, aggiunge quindi "*-*-*-*-*", assegna alla nuova stringa il nome lungaStringa$ e quindi cancella quella precedente per fare spazio per altre stringhe. Durante il processo QBasic necessita di uno spazio sufficiente alle due copie di lungaStringa$; si supponga di avere 20.000 byte di spazio libero per la stringa e che lungaStringa$ abbia una lunghezza di 10.000 byte. Se si cerca di concatenare "*-*-*-*-*"a questa variabile, il compilatore restituisce il messaggio di errore: "Spazio stringhe esaurito", dato che QBasic effettua un'altra copia di lungaStringa$, non lasciando spazio sufficiente per la copia di "*-*-*-*-*".

Esempio

Il programma di seguito riportato utilizza l'operatore + per combinare le tre parti che compongono il nome di un impiegato.

```
CLS
LINE INPUT "Inserire il nome dell'impiegato:                    "; nome$
globNome$ = nome$ + " "

LINE INPUT "Inserire il cognome dell'impiegato:                 "; cognome$
globNome$ = globNome$ + cognome$

INPUT "Inserire l'ammontare dello stipendio dell'impiegato:     "; ammontare#

PRINT
PRINT "Pagare a "; globNome$; TAB(60);
PRINT USING "L. ##,###,###"; ammontare#
END
```

EQV

Si consulti anche

AND, IF, IMP, MOD, NOT, OR, XOR

Scopo

EQV è un operatore sia logico che aritmetico. Come operatore logico, restituisce il vero soltanto se entrambi gli operandi sono veri o falsi. Come operatore aritmetico, esegue operazioni EQV con i bit corrispondenti di due valori.

Sintassi

```
IF op1 EQV op2 THEN...
```

oppure

```
risultato% = op1 EQV op2
```

Gli operandi op1 e op2 nel primo caso possono essere espressioni logiche oppure valori di qualsiasi tipo numerico (intero, intero lungo, a precisione semplice o a precisione doppia); nel secondo caso possono essere di qualsiasi tipo numerico, ma il risultato è sempre un valore di tipo intero o intero lungo.

risultato% è la variabile che contiene il risultato di un'operazione aritmetica EQV.

Utilizzo

```
IF importo1# > 100# EQV importo2# > 2000# THEN
   PRINT "Gli importi si trovano nell'intervallo corretto"
END IF
```

Se importo1# è maggiore di 100 e importo2# è maggiore di 2000, l'espressione è vera; se importo1# è inferiore a 100 e importo2# è inferiore a 2000, l'espressione è ancora vera, dato che essa risulta falsa soltanto quando solo un valore è vero e l'altro è falso.

Descrizione

L'operatore logico EQV può essere utilizzato in un'espressione logica per determinare se entrambi gli operandi sono veri o falsi. Ad esempio l'espressione:

```
var1 > 2 EQV var2 < 4
```

è vera soltanto quando le espressioni var1>2 e var2<4 sono entrambe vere o entrambe false.

L'operatore EQV può anche essere utilizzato in modo aritmetico per eseguire un confronto bit per bit degli operandi. Se i bit corrispondenti negli operandi sono entrambi 1 o 0, il risultato è 1; in caso contrario, il risultato è 0. Si consulti la tabella di verità riportata nella Tabella 3.4 per ulteriori dettagli.

Tabella 3.4 La tabella di verità EQV

p	q	p EQV q
1	1	1
1	0	0
0	1	0
0	0	1

Questo operatore viene generalmente utilizzato con maggiore frequenza nelle espressioni logiche, rispetto a quelle aritmetiche.

L'operatore EQV accetta operandi a precisione semplice e a precisione doppia, convertendoli però in interi o interi lunghi prima di elaborarli; il risultato è sempre un valore di tipo intero o intero lungo.

Errori

Gli operandi di un'operazione EQV possono essere di qualsiasi tipo numerico (intero, intero lungo, a precisione semplice, a precisione doppia), compresi nell'intervallo da -2147483648 a 2147483647; in caso contrario, QBasic visualizza il messaggio di errore: "Overflow".

Esempio

L'esempio di seguito riportato illustra l'utilizzo dell'operatore EQV per determinare l'idoneità di una persona a ottenere un prestito.

```
CLS
PRINT "Applicazione PRESTITI"
PRINT

INPUT "Ammontare del prestito desiderato"; ammontare#
INPUT "Qual è lo stipendio annuale del richiedente"; stipendio#

PRINT
IF ammontare# > 10000000 EQV stipendio# > 20000000 THEN
    PRINT "Poiché l'ammontare del prestito richiesto è"
    PRINT "appropriato per lo stipendio del richiedente,"
    PRINT "il prestito di L."; ammontare#; "è approvato."
ELSE
    PRINT "Il richiedente non soddisfa le nostre condizioni; ";
    PRINT "il prestito è negato."
END IF
```

IF

Si consulti anche

DO...LOOP, ON...GOSUB, ON...GOTO, SELECT CASE

Scopo

L'istruzione IF utilizza il valore di un'espressione per controllare l'ordine di esecuzione delle istruzioni. L'espressione in un'istruzione IF può essere un confronto logico; utilizzando gli operatori logici AND, OR, NOT, XOR, EQV e IMP è possibile combinare diversi confronti.

Sintassi

Monoriga:

```
IF espressione THEN vero ELSE falso
```

A blocchi:

```
IF espressione1 THEN
   [vero]
[ELSEIF espressione2 THEN
   [vero]]
.
.
.
[ELSEIF espressionex THEN
   [vero]]
[ELSE
   [falso]]
END IF
```

espressione è un'espressione che dà come risultato il vero o il falso. L'espressione può utilizzare gli operatori relazionali di QBasic (=, <>, <, >, >=, <=) e gli operatori

logici (AND, OR, NOT, XOR, EQV, IMP) per produrre un valore vero o falso. Il programma può rappresentare un valore falso con uno 0 e uno vero con un valore diverso da zero, quindi espressione può anche essere un'espressione di tipo intero.

vero è un gruppo di istruzioni che deve essere eseguito se espressione è un valore vero. Nell'istruzione IF monoriga, le istruzioni multiple devono essere sulla stessa riga, separate da due punti. Nella forma a blocchi, le istruzioni multiple possono essere sulla stessa riga (separate da due punti) oppure su righe diverse.

Le istruzioni successive alla clausola ELSEIF, che sono presenti soltanto nell'istruzione IF a blocchi, vengono eseguite soltanto se espressione1 è falsa ed espressione2 è vera. ELSEIF consente di controllare molte condizioni diverse all'interno di un'istruzione IF monoriga.

QBasic esegue falso, ovvero il gruppo di istruzioni che segue la parola chiave ELSE, se tutte le espressioni precedenti IF ed ELSEIF sono false. La clausola ELSE è una clausola "contenitore", se nessuna delle espressioni viene soddisfatta, QBasic esegue le istruzioni successive alla parola chiave ELSE.

Utilizzo

```
IF eta% < 18 THEN stato$ = "Minore"
```

Se il valore della variabile eta% è inferiore a 18, l'istruzione imposta stato$ a "Minore".

```
IF dimensioneMem& < 250 THEN
   PRINT "Memoria non sufficiente"
   BEEP
   SYSTEM
END IF
```

Il programma precedente stampa un messaggio, emette un suono acustico e quindi esce se il valore della variabile dimensioneMem& è inferiore a 250.

```
IF risposta$ = "S" THEN 10 ELSE 20

IF risposta$ = "S" THEN GOTO 10 ELSE GOTO 20

IF risposta$ = "S" THEN
   GOTO 10
ELSE
   GOTO 20
END IF
```

Queste tre istruzioni sono equivalenti, dato che tutte consentono di passare alla riga 10 se il valore della variabile risposta$ è uguale a "S", e alla riga 20 in caso contrario.

```
IF eta% >= 18 AND patente$ = "NO" THEN
   PRINT "Fai l'esame per la patente ...e guida con prudenza."
   GOTO PrenderePatenteGuida
ELSEIF eta% > 18 AND patente$ = "S" THEN
   PRINT "Rinnova la patente"
   GOTO RinnovoPatente
ELSE
   PRINT "Mi spiace, ma devi aspettare..."
   GOTO TroppoGiovane
END IF
```

Verifica il valore della variabile eta% e di patente$ e stampa il messaggio che corrisponde alla combinazione di questi valori.

Descrizione

L'istruzione IF di QBasic consente di verificare se una determinata condizione è stata soddisfatta; in caso positivo QBasic esegue le istruzioni indicate, se invece la condizione non viene soddisfatta, vengono eseguite istruzioni diverse. Questa istruzione fornisce il livello più basso di capacità decisionali in QBasic; al contrario dei cicli di controllo del flusso, le istruzioni IF vengono eseguite soltanto una volta.

Quando l'istruzione IF trasferisce il controllo a un'altra riga numerata nel programma, è possibile omettere la parola chiave GOTO; se la riga ha un'etichetta, è tuttavia necessario utilizzare quest'ultima istruzione. Le istruzioni di seguito riportate illustrano un utilizzo corretto dell'istruzione IF.

```
IF importoDov# > 1000.0 THEN GOTO 10000
IF importoDov# > 1000.0 THEN 10000
```

È bene utilizzare l'istruzione IF monoriga, quando la serie di istruzioni che vengono eseguite nel caso l'espressione sia vera comprende soltanto una coppia di istruzioni. In questo caso l'istruzione IF monoriga risulta più concisa e semplice da leggere rispetto a quella a blocchi, come riportato nel seguente esempio.

```
IF importoDov# > 1000.0 THEN GOTO 10000

IF importoDov# > 1000.0 THEN
   GOTO 10000
END IF
```

L'istruzione IF a blocchi consente di inserire tutte le istruzioni necessarie, senza doverle immettere (separate da due punti) su una sola riga.

QBasic valuta l'istruzione IF partendo dall'alto e proseguendo verso il basso. Ogni istruzione IF a blocchi deve terminare con una clausola END IF corrispondente. Ogniqualvolta il valore di un'espressione è vero, QBasic esegue le istruzioni della riga successiva alla parola chiave THEN; se invece le espressioni corrispondenti alla clausola IF e a tutte le clausole ELSEIF sono false, QBasic esegue le istruzioni successive alla parola chiave ELSE. È necessario tenere presente che l'inserimento di istruzioni nelle clausole IF, ELSEIF e ELSE è facoltativo: se vengono omesse, QBasic non esegue alcuna operazione se l'espressione è vera. Ad esempio, l'istruzione IF a blocchi di seguito riportata è corretta.

```
IF eta% >= 18 THEN
ELSE
   PRINT "Sei in arresto!"
END IF
```

Si noti inoltre che le clausole ELSEIF e ELSE sono facoltative: se l'espressione IF è falsa e non vi sono clausole ELSEIF o ELSE, QBasic prosegue l'esecuzione in corrispondenza della prima istruzione successiva alla clausola END IF.

Un'istruzione IF a blocchi può comprendere altre istruzioni strutturate a blocchi, quali altre istruzioni IF e DO...LOOP, mentre tutto ciò non è possibile con la forma monoriga. Le clausole IF, ELSEIF, ELSE ed END IF dell'istruzione IF a blocchi devono essere le prime istruzioni su ogni riga, anche se è possibile che siano precedute da un numero di riga o da un'etichetta, nel caso si dovesse trasferire il controllo a queste istruzioni da un'altra parte del programma.

Dato che QBasic esamina la parte di codice successiva alla clausola THEN per determinare se IF è un'istruzione monoriga o a blocchi, in un'istruzione IF strutturata a blocchi non è possibile aggiungere alcunché, tranne un commento, dopo la parola chiave THEN.

Errori

Se si utilizza un'espressione numerica in un'istruzione IF, essa deve avere un valore compreso nell'intervallo di un intero lungo, ovvero da -2147483648 a 2147483647. Se viene utilizzata un'espressione in virgola mobile che supera questo intervallo, QBasic visualizza il messaggio di errore: "Overflow".

Suggerimenti

Per verificare se il valore di una variabile è 0, è sufficiente utilizzare l'istruzione:

```
IF NOT variabile%
```

se variabile% è 0, allora l'espressione NOT variabile% è vera. La semplice espressione variabile% nell'istruzione di seguito riportata è vera se variabile% è uguale a qualsiasi valore intero, tranne 0.

```
IF variabile%
```

Talvolta le operazioni in virgola mobile producono risultati imprecisi; ne consegue che valori in virgola mobile che sembrano uguali potrebbero risultare leggermente diversi una volta stampati. Se i numeri in virgola mobile vengono arrotondati, prima di confrontarli, le differenze non influenzano il confronto.

Esempio

L'esempio di seguito riportato causa l'emissione di un suono (da bassa ad alta frequenza) e la visualizzazione della frequenza e della gamma dei toni.

```
CLS
FOR frequenza% = 100 TO 1000 STEP 10
    IF RND <= .5 THEN durata% = 1 ELSE durata% = 2 'sceglie una durata casuale

    SOUND frequenza%, durata%
    LOCATE 1, 1
    PRINT USING "##### "; frequenza%;

    IF frequenza% < 200 THEN
        PRINT "Frequenza molto bassa "
    ELSEIF frequenza% < 400 THEN
        PRINT "Frequenza bassa       "
    ELSEIF frequenza% < 600 THEN
        PRINT "Frequenza media       "
    ELSEIF frequenza% < 800 THEN
        PRINT "Frequenza alta        "
    ELSE
        PRINT "Frequenza molto alta  "
    END IF
NEXT
END
```

IMP

Si consulti anche

AND, EVQ, IF, MOD, NOT, OR, XOR

Scopo

IMP è un operatore sia logico che aritmetico. Come operatore logico, restituisce il falso soltanto se il primo operando è vero e il secondo è falso. Come operatore aritmetico, esegue operazioni IMP con i bit corrispondenti di due valori.

Sintassi

```
IF op1 IMP op2 THEN...
```

oppure

```
risultato% = op1 IMP op2
```

Gli operandi op1 e op2 nel primo caso possono essere espressioni logiche oppure valori di qualsiasi tipo numerico (intero, intero lungo, a precisione semplice o a precisione doppia); nel secondo caso possono essere di qualsiasi tipo numerico, ma il risultato è sempre un valore di tipo intero o intero lungo.

risultato% è la variabile che contiene il risultato di un'operazione aritmetica IMP.

Utilizzo

```
IF importo > 1000 IMP "OcchialiProtezione$ = "SI" THEN
   PRINT "Pronti a procedere"
END IF
```

L'espressione presente nell'istruzione IF è falsa soltanto se importo è maggiore di 1000 e OcchialiProtezione$ non è uguale a "SI".

Descrizione

L'operatore logico IMP può essere utilizzato in un'espressione logica per verificare l'implicazione: l'intera espressione è vera se il valore del primo operando implica il valore del secondo. Un'espressione che utilizza l'operatore IMP è vera, tranne quando il primo operando è vero e il secondo è falso. Ad esempio:

```
var1 > 2 IMP var2 < 4
```

è falsa soltanto quando l'espressione var1 > 2 è vera e var2 < 4 è falsa.

L'operatore IMP può anche essere utilizzato in modo aritmetico per eseguire un confronto bit per bit degli operandi. Se un bit in op1 è 1 e il bit corrispondente in op2 è 0, il risultato è 0; in caso contrario, il risultato è 1. Si consulti la tabella di verità riportata nella Tabella 3.5 per ulteriori dettagli.

Generalmente questo operatore viene utilizzato con maggiore frequenza nelle espressioni logiche, rispetto a quelle aritmetiche.

Tabella 3.5 La tabella di verità IMP

p	q	p IMP q
1	1	1
1	0	0
0	1	1
0	0	1

L'operatore IMP accetta operandi a precisione semplice e a precisione doppia, convertendoli però in interi o interi lunghi prima di elaborarli; il risultato è sempre un valore di tipo intero o intero lungo.

Errori

Gli operandi di un'operazione IMP possono essere di qualsiasi tipo numerico (intero, intero lungo, a precisione semplice, a precisione doppia), compresi nell'intervallo da -2147483648 a 2147483647; in caso contrario, QBasic visualizza il messaggio di errore: "Overflow".

Esempio

L'esempio di seguito riportato illustra l'utilizzo dell'operatore IMP per determinare l'idoneità di una persona a ottenere un mutuo.

```
CLS
PRINT "Applicazione PRESTITI": PRINT

INPUT "Ammontare del prestito desiderato"; ammontare#
INPUT "Il richiedente è impiegato a tempo pieno (S/N)"; tempoPieno$

PRINT
IF ammontare# > 1000000 IMP UCASE$(tempoPieno$) = "S" THEN
    PRINT "Poiché l'ammontare del prestito è appropriato";
    PRINT "allo status lavorativo del richiedente,"
    PRINT "il prestito di "; ammontare#; "è approvato."
ELSE
    PRINT "Il richiedente non soddisfa le nostre condizioni; ";
    PRINT "il prestito è negato."
END IF
```

MOD

Scopo

MOD è un operatore aritmetico che restituisce il resto di una divisione tra due interi; può essere utilizzato, ad esempio, per iniziare una nuova pagina di un prospetto ogni 55 righe.

Sintassi

```
resto% = op1 MOD op2
```

Gli operandi op1 e op2 possono essere valori di qualsiasi tipo numerico (intero, intero lungo, a precisione semplice o a precisione doppia), ma il risultato è sempre un valore di tipo intero o intero lungo.

resto% è la variabile che contiene il risultato di un'operazione MOD.

Utilizzo

```
IF riga% MOD 55 = 0 THEN
   LPRINT CHR$(12);
END IF
```

Se l'operazione MOD non genera alcun resto, il programma invia alla stampante un carattere di salto pagina.

Descrizione

L'operatore MOD divide due operandi e restituisce il resto della divisione. Prima di eseguire la divisione, gli operandi vengono arrotondati a interi; anche il risultato della divisione viene arrontondato a intero, prima che venga calcolato il resto. Il risultato di un'operazione MOD è sempre un valore di tipo intero o intero lungo.

Errori

Gli operandi di un'operazione MOD possono essere di tipo numerico (intero, intero lungo, a precisione semplice, a precisione doppia), compresi nell'intervallo da -2147483648 a 2147483647; in caso contrario, QBasic visualizza il messaggio di errore: "Overflow".

Esempio

L'esempio di seguito riportato utilizza l'operatore MOD per interrompere la visualizzazione dell'output alla fine di ogni schermata.

```
x = 1
DO
  PRINT TAB(x MOD 55); "Guida di riferimento Qbasic"

  IF x MOD 23 = 0 THEN
       PRINT "Premi [Esc] per terminare il programma o qualunque tasto per
       continuare ...";
       DO
            attesa$ = INKEY$
       LOOP WHILE attesa$ = ""
       IF attesa$ = CHR$(27) THEN EXIT DO
  END IF
  x = x + 1
LOOP
```

NOT

Si consulti anche

AND, EQV, IF, IMP, MOD, OR, XOR

Scopo

NOT è un operatore sia logico che aritmetico. Come operatore logico restituisce il vero se l'operando è falso. Come operatore aritmetico, inverte i valori dei bit del proprio operando. L'operatore NOT deve essere utilizzato per invertire il valore di verità di una variabile.

Sintassi

```
IF NOT op THEN...
```

oppure

```
risultato% = NOT op
```

L'operando op nel primo caso può essere un'espressione logica oppure un valore di qualsiasi tipo numerico (intero, intero lungo, a precisione semplice o a precisione doppia); nel secondo caso può essere di qualsiasi tipo numerico, ma il risultato è sempre un valore di tipo intero o intero lungo.

Utilizzo

```
IF NOT stampantePronta% THEN
   PRINT "Inserire la carta nella stampante e premere Invio"
END IF
```

Questo esempio visualizza un messaggio quando la stampante non è pronta. La variabile stampantePronta% rappresenta lo stato della stampante e assume il valore vero quando la stampante è pronta. Dato che è necessario sapere quando la stampante non è pronta, è sufficiente applicare l'operatore NOT alla variabile stampantePronta%.

Descrizione

L'operatore logico NOT può essere utilizzato in un'espressione logica per determinare se un operando è falso. Ad esempio l'espressione:

```
NOT var1 > 2
```

è vera soltanto quando l'espressione var1>2 è falsa.

L'operatore NOT può anche essere utilizzato in modo aritmetico per invertire i bit dell'operando. Se un bit è 1, il bit risultante è 0; se invece è 0, il risultato è 1. Si consulti la tabella di verità riportata nella Tabella 3.6 per ulteriori dettagli.

In base al metodo utilizzato da QBasic per memorizzare gli interi e gli interi lunghi, l'utilizzo dell'operatore NOT è analogo all'operazione di aggiungere 1 al valore e renderlo negativo.

L'operatore NOT accetta operandi a precisione semplice e a precisione doppia, convertendoli in interi o interi lunghi prima di elaborarli; il risultato è sempre un intero o un intero lungo.

Errori

L'operando di un'operazione NOT può essere di tipo numerico (intero, intero lungo, a precisione semplice, a precisione doppia), compreso nell'intervallo da -2147483648 a 2147483647; in caso contrario, QBasic visualizza il messaggio: "Overflow".

Tabella 3.6 La tabella di verità NOT

p	NOT p
1	0
0	1

Esempio

L'esempio di seguito riportato utilizza l'operatore NOT per invertire il valore di una funzione definita dall'utente Ok%, in modo che il programma non termini fino a quando la funzione non restituisce il vero.

```
DECLARE FUNCTION Ok% (richiesta$)
CONST Falso = 0, Vero = NOT Falso

CLS
DO WHILE NOT Ok("Pronto a terminare il programma.  ")  'chiama la funzione Ok
LOOP                               'cicla mentre la funzione Ok ritorna vero

SYSTEM

'La funzione Ok ritorna VERO se l'utente inserisce "S"


FUNCTION Ok% (richiesta$) STATIC
    LOCATE , , 1                     'visualizza il cursore
    PRINT richiesta$; "Va bene?  ";

    DO
       tastiera$ = UCASE$(INKEY$)  'ottiene un carattere
    LOOP WHILE NOT (tastiera$ = "N" OR tastiera$ = "S")

    SELECT CASE tastiera$
        CASE "N"
            PRINT "No"
            Ok% = Falso

        CASE "S"
            PRINT "Si"
            Ok% = Vero
        END SELECT
END FUNCTION
```

OR

Si consulti anche

AND, EQV, IF, IMP, MOD, NOT, XOR

Scopo

OR è un operatore sia logico che aritmetico. Come operatore logico, restituisce il vero soltanto se uno o entrambi gli operandi sono veri. Come operatore aritmetico, esegue operazioni OR con i bit corrispondenti di due valori.

Sintassi

```
IF op1 OR op2 THEN...
```

oppure

```
risultato% = op1 OR op2
```

Gli operandi op1 e op2 nel primo caso possono essere espressioni logiche oppure valori di qualsiasi tipo numerico (intero, intero lungo, a precisione semplice o a precisione doppia); nel secondo caso possono essere di qualsiasi tipo numerico, ma il risultato è sempre un valore di tipo intero o intero lungo.

risultato% è la variabile che contiene il risultato di un'operazione aritmetica OR.

Utilizzo

```
IF ricco% OR famoso% THEN
   PRINT "Perfetto...sei sposato?"
END IF
```

Se una o entrambe le variabili ricco% e famoso% sono vere, viene visualizzato un messaggio.

Descrizione

L'operatore logico OR può essere utilizzato in un'espressione logica per determinare se uno o entrambi gli operandi sono veri. Ad esempio l'espressione:

```
var1 > 2 OR var2 < 4
```

è vera quando l'espressione var1>2 è vera, oppure quando sono vere entrambe.

Questa operazione è di tipo "o inclusivo", dà come risultato il vero se uno o entrambi gli operandi sono veri. Se fosse necessario controllare se soltanto uno e non entrambi gli operandi sono veri, è possibile utilizzare XOR.

L'operatore OR può anche essere utilizzato in modo aritmetico per eseguire un confronto bit per bit degli operandi. Se uno o entrambi i bit corrispondenti negli operandi è 1, il bit risultante è 1; in caso contrario il risultato è 0. Si consulti la tabella di verità riportata nella Tabella 3.7 per ulteriori dettagli.

Nella Figura 3.2 è illustrato l'utilizzo dell'operatore aritmetico OR.

L'operatore OR accetta operandi a precisione semplice e a precisione doppia, convertendoli in interi o interi lunghi prima di elaborarli; il risultato è sempre un intero o un intero lungo.

Errori

Gli operandi di un'operazione OR possono essere di tipo numerico (intero, intero lungo, a precisione semplice, a precisione doppia), compresi nell'intervallo da -2147483648 a 2147483647; in caso contrario, QBasic visualizza il messaggio di errore: "Overflow".

Esempio

L'esempio di seguito riportato illustra il metodo di utilizzo dell'operatore OR per determinare l'idoneità di una persona a ottenere un prestito.

Tabella 3.7 La tabella di verità OR

p	q	p OR q
1	1	1
1	0	1
0	1	1
0	0	0

```
L'istruzione aritmetica OR

      262 OR 39

viene calcolata come segue:

      262 decimale = 100000110 binario
   OR  39 decimale = 000100111 binario
      ------------------------------
      295 decimale = 100100111 binario
                     |  |  |||
            Uno o entrambi i bit in
             queste colonne sono 1
```

Figura 3.2 OR aritmetico

```
CLS
PRINT "Applicazione PRESTITI": PRINT

INPUT "Ammontare del prestito desiderato"; ammontare#
INPUT "Il richiedente è impiegato a tempo pieno (S/N)"; tempoPieno$
INPUT "Il richiedente ha sufficienti garanzie (S/N)"; garanzie$

PRINT
IF UCASE$(tempoPieno$) = "S" AND UCASE$(garanzie$) = "S" THEN
  PRINT "Poiché il richiedente domanda un prestito ragionevolmete garantito,";
  PRINT "il prestito di L."; ammontare#; "è approvato."
ELSE
  PRINT "Il richiedente non soddisfa le nostre condizioni; ";
  PRINT "il prestito è negato."
END IF
```

SELECT CASE

Si consulti anche

IF

Scopo

L'istruzione SELECT CASE fornisce un metodo molto chiaro per confrontare un'espressione con diverse costanti, eseguendo quindi le istruzioni in base al confronto effettuato. Questa istruzione rappresenta un'alternativa di più facile lettura e maggiormente strutturata rispetto alle complesse istruzioni IF...ELSE.

Sintassi

```
SELECT CASE espressione
   CASE comparazione [,comparazione...]
      [bloccoistruzioni]
   [CASE comparazione [,comparazione...]
      [bloccoistruzioni]]
.
.
.
[CASE ELSE
[bloccoistruzioni]]
END SELECT
```

espressione è una qualsiasi espressione numerica o stringa.

comparazione è un'espressione che rappresenta un intervallo di valori in base ai quali viene confrontata espressione.

comparazione deve essere di una delle seguenti forme:

espressione	Una singola espressione
minespressione TO maxespressione	Un intervallo di valori
IS espressione di relazione	Un intervallo di comparazione

bloccoistruzioni indica una o più istruzioni di QBasic.

Utilizzo

```
SELECT CASE menu&
   CASE 1
      AggiuntaClienti
   CASE 2
      AggiuntaTransazioni
END SELECT
```

Se menu& è uguale a 1 il programma esegue il sottoprogramma AggiuntaClienti, se menu& è uguale a 2 il programma esegue il sottoprogramma AggiuntaTransazioni.

```
SELECT CASE ok$
   CASE "S", "s"
      Effettuare
   CASE "N", "n"
      Annullare
END SELECT
```

Se la variabile ok$ è uguale a "S" oppure a "s", il programma esegue il sottoprogramma Effettuare; se invece è uguale a "N" o "n", il programma esegue il sottoprogramma Annullare.

```
SELECT CASE anno%
   CASE 1901 TO 1930
      PRINT "All'inizio del ventesimo secolo..."
   CASE 1931 TO 1960
```

```
        PRINT "Verso la metà del ventesimo secolo..."
    CASE IS > 1961
        PRINT "Dopo la metà e verso la fine del ventesimo secolo..."
SELECT CASE
```

Se la variabile anno% ha un valore compreso tra il 1901 e il 1930, tra il 1931 e il 1960 oppure è maggiore di 1960, viene visualizzato il messaggio corretto.

Descrizione

L'istruzione IF (esaminata in precedenza) si colloca al livello più basso delle strutture decisionali di QBasic. IF opera in base a confronti logici, mentre SELECT CASE è più efficace con intervalli e confronti relazionali. Dato che questa istruzione accetta gli operatori relazionali, è possibile effettuare qualsiasi tipo di confronto con SELECT CASE oppure con una combinazione delle istruzioni IF, ELSEIF e ELSE.

L'espressione presente in una clausola SELECT CASE può essere un intero, un numero in virgola mobile oppure un'espressione stringa. Ad esempio, nella clausola di seguito riportata il valore dell'espressione LOG(e) viene confrontato con le costanti delle clausole CASE:

```
SELECT CASE LOG(e)
```

SELECT CASE confronta il valore dell'espressione con i valori o l'intervallo di valori nelle clausole CASE. Se l'espressione corrisponde al valore o all'intervallo di valori, le istruzioni di quella determinata clausola CASE vengono eseguite. Dopo l'esecuzione di un gruppo di istruzioni, il controllo passa alla prima istruzione successiva alla clausola END SELECT.

Di seguito sono riportati alcuni esempi di valori corretti per quanto riguarda l'espressione di confronto. Una singola espressione:

CASE 3.141592653# *arc#	Corrisponde al valore di π volte arc#
CASE 1000&	Corrisponde all'intero lungo 1000
CASE NomePaziente$	Corrisponde alla variabile stringa NomePaziente$

Un intervallo di valori (minespressione TO maxespressione, dove il primo deve essere inferiore rispetto al secondo, numericamente o rispetto all'ordine ascendente ASCII):

```
CASE 1 TO 10       Corrisponde ai valori nell'intervallo da 1.0 a 10.0
CASE "A" TO "Z"    Corrisponde a tutte le lettere maiuscole
```

Un confronto all'interno di un intervallo:

CASE IS=1	Corrisponde ai numeri uguali a 1
CASE 1	Corrisponde ai numeri uguali a 1
CASE IS<=10	Corrisponde ai numeri minori di o uguali a 10
CASE IS<="Z"	Corrisponde a stringhe minori di o uguali a Z

La parola chiave IS fornisce all'istruzione SELECT CASE tutta la funzionalità dell'istruzione IF, pemettendo di utilizzare gli operatori relazionali di QBasic (=, <>, <, >, >=, <=). Si noti che una clausola CASE che utilizza il segno di uguale equivale a una clausola CASE che utilizza una singola espressione.

Se nessuno dei valori presenti nelle istruzioni CASE corrisponde all'espressione,

QBasic esegue le istruzioni della clausola CASE ELSE (se ne esiste una).

In ogni clausola CASE è possibile effettuare diversi confronti (separati da virgole); in questo caso, QBasic esegue le istruzioni associate se uno qualsiasi dei confronti viene soddisfatto. Le virgole funzionano come l'operatore logico OR nell'istruzione IF. Non è possibile simulare l'operatore logico AND.

Suggerimenti

Dato che con SELECT CASE non è possibile utilizzare gli operatori logici, l'istruzione IF rappresenta una scelta migliore quando il programma deve effettuare confronti logici (utilizzando gli operatori logici AND, OR, NOT, XOR, EQV e IMP).

Talvolta le operazioni in virgola mobile producono risultati imprecisi; ne consegue che valori in virgola mobile che sembrano uguali, potrebbero risultare leggermente diversi una volta stampati. Se i numeri in virgola mobile vengono arrotondati, prima di confrontarli, le differenze non influenzano il confronto.

Esempio

L'esempio di seguito riportato utilizza l'istruzione SELECT CASE per determinare quanti caratteri in una frase sono vocali, quanti sono consonanti e quanti segni di punteggiatura.

```
CLS
LINE INPUT "Inserire una frase: "; frase$

vocali = 0: consonanti = 0: punteggiatura = 0  'inizializza le variabili
                                                'contatore

FOR X = 1 TO LEN(frase$)                        'cicla attraverso la stringa
    carattere$ = UCASE$(MID$(frase$, X, 1))     'ottiene un singolo carattere

    SELECT CASE carattere$
        CASE "A", "E", "I", "O", "U"            'controllo per le vocali
            vocali = vocali + 1
        CASE "A" TO "Z"                         'consonanti
            consonanti = consonanti + 1         'include le vocali, ma queste
                                                'sono conteggiate prima
        CASE ELSE
            punteggiatura = punteggiatura + 1   'tutti gli altri caratteri sono
                                                'segni di punteggiatura
    END SELECT
NEXT

'stampa i totali
PRINT "Nella frase contenente"; LEN(frase$); "caratteri,"
PRINT "ci sono"; vocali; "vocali, "; consonanti; "consonanti e";
PRINT punteggiatura; "segni di punteggiatura"
```

XOR

Si consulti anche

AND, EQV, IF, IMP, MOD, NOT, XOR

Scopo

XOR è un operatore sia logico che aritmetico. Come operatore logico restituisce il

vero se uno, ma non entrambi gli operandi sono veri. Questo operatore deve essere utilizzato per assicurarsi che soltanto una variabile sia vera, impedendo in questo modo conflitti di opzioni. Come operatore aritmetico, esegue operazioni XOR con i bit corrispondenti di due valori.

Sintassi

```
IF op1 XOR op2 THEN...
```

oppure

```
risultato% = op1 XOR op2
```

Gli operandi op1 e op2 nel primo caso possono essere espressioni logiche oppure valori di qualsiasi tipo numerico (intero, intero lungo, a precisione semplice o a precisione doppia); nel secondo caso possono essere di qualsiasi tipo numerico, ma il risultato è sempre un valore di tipo intero o intero lungo.

risultato% è la variabile che contiene il risultato di un'operazione aritmetica XOR.

Utilizzo

```
IF discofisso% XOR dischetto% THEN
   PRINT "Ora, è possibile installare il programma"
END IF
```

Se una delle variabili discofisso% e dischetto% è vera (ma non entrambe), viene visualizzato il messaggio.

Descrizione

L'operatore logico XOR può essere utilizzato in un'espressione logica per determinare se uno degli operandi è vero (ma non entrambi). Ad esempio l'espressione:

```
var1 > 2 XOR var2 < 4
```

è vera soltanto quando l'espressione var1 > 2 è vera e var2 < 4 è falsa, oppure quando l'espressione var1 > 2 è falsa e var2 < 4 è vera.

L'operatore XOR può anche essere utilizzato in modo aritmetico per eseguire un confronto bit per bit degli operandi. Se i bit corrispondenti negli operandi hanno valori diversi, il bit risultante è 1; in caso contrario il risultato è 0. Si consulti la tabella di verità riportata nella Tabella 3.8 per ulteriori dettagli.

Nella Figura 3.3 è illustrato l'utilizzo dell'operatore aritmetico XOR.

L'operatore XOR accetta operandi a precisione semplice e a precisione doppia, convertendoli in interi o interi lunghi prima di elaborarli; il risultato è sempre un intero o un intero lungo.

Errori

Gli operandi di un'operazione XOR possono essere di tipo numerico (intero, intero lungo, a precisione semplice, a precisione doppia), compresi nell'intervallo da -2147483648 a 2147483647; in caso contrario, QBasic visualizza il messaggio di errore: "Overflow".

Tabella 3.8 La tabella di verità XOR

p	q	p XOR q
1	1	0
1	0	1
0	1	1
0	0	0

```
L'istruzione aritmetica XOR

      262 XOR 39

viene calcolata come segue:

      262 decimale = 100000110 binario
   XOR 39 decimale = 000100111 binario
      ________________________________

      289 decimale = 100100001 binario
                        |  |   |
      L'uno o l'altro (ma non entrambi) dei bit
              in queste colonne sono 1
```

Figura 3.3 XOR aritmetico

Esempio

L'esempio di seguito riportato illustra l'utilizzo dell'operatore XOR in una clausola UNTIL di un'istruzione DO...LOOP per eseguire un ciclo fino a quando un utente seleziona soltanto uno dei due tipi di prestito.

```
CLS
PRINT "Tipo di prestito": PRINT

DO
    DO
        INPUT "Desideri un prestito a tasso di interesse fisso (S/N)"; fisso$
        INPUT "Desideri un prestito a tasso di interesse variabile (S/N)";
        variabile$
        PRINT
    LOOP UNTIL UCASE$(fisso$) = "S" XOR UCASE$(variabile$) = "S"

    IF UCASE$(fisso$) = "S" THEN
        PRINT "Hai scelto un prestito a tasso di interesse fisso."
    ELSE
        PRINT "Hai scelto un prestito a tasso di interesse variabile."
    END IF
    INPUT "Hai cambiato idea (S/N)"; cambiamento$
LOOP UNTIL UCASE$(cambiamento$) = "N"
```

Capitolo 4

Procedure

INTRODUZIONE

Dato che le versioni precedenti di Basic non dispongono di caratteristiche strutturate, se i programmatori non realizzano i propri programmi con attenzione, rischiano di renderli difficili da leggere e da gestire. Senza dimenticare che i programmi che utilizzano le versioni precedenti di Basic presentano dei limiti per quanto riguarda le dimensioni dei file sorgenti che l'interprete è in grado di elaborare.

QBasic non presenta nessuno di questi problemi, dato che offre tutte le caratteristiche strutturate peculiari di linguaggi di programmazione quali C e Pascal. È possibile definire funzioni e sottoprogrammi, passando le variabili e i valori richiesti. È inoltre possibile controllare la visibilità delle variabili, condividendole con altre procedure soltanto se questo è necessario.

In questo capitolo vengono esaminate in modo dettagliato le caratteristiche strutturate di QBasic. Nella Tabella 4.1 sono riportate in ordine alfabetico le istruzioni e le funzioni procedurali di QBasic.

DIM, SHARED e COMMON

Le variabili e gli array possono essere dimensionati nel programma principale. Se si desidera che siano disponibili per tutte le procedure di un programma, è necessario utilizzare l'istruzione DIM con la parola chiave SHARED; senza quest'ultima, le variabili dimensionate sono visibili ai sottoprogrammi e alle funzioni soltanto se vengono specificate in un'istruzione SHARED in ogni procedura che le utilizza. L'istruzione DIM con la parola chiave SHARED non rende le variabili disponibili per le procedure presenti in altri programmi; questa operazione può essere eseguita soltanto con l'istruzione COMMON.

Le variabili specificate mediante l'istruzione COMMON non sono visibili ai sottoprogrammi e alle funzioni del programma. Per renderle visibili, è necessario

Tabella 4.1 Le istruzioni e le funzioni relative alle procedure

Istruzione o funzione	Descrizione
CALL	Trasferisce il controllo a un sottoprogramma QBasic
CHAIN	Trasferisce il controllo a un altro programma
COMMON	Dichiara le variabili che possono essere passate a un programma concatenato separato
DECLARE	Definisce l'elenco dei nomi e dei parametri di una procedura specificata da un'istruzione SUB o FUNCTION
DEF FN	Specifica una funzione definita dall'utente compatibile con quelle scritte con le versioni precedenti di Basic
FUNCTION	Definisce una funzione QBasic
GOSUB...RETURN	Passa il controllo a una subroutine
SHARED	Specifica le variabili che possono essere condivise dalle funzioni, dai sottoprogrammi e dalle istruzioni nel codice a livello del modulo
STATIC	Specifica le variabili locali in una funzione o in un sottoprogramma che conservano i valori tra le chiamate alla funzione o al sottoprogramma
SUB	Definisce un sottoprogramma

inserire la parola chiave SHARED in un'istruzione COMMON oppure includere un'istruzione SHARED separata in ogni procedura che ha la necessità di accedere alle variabili specificate dall'istruzione COMMON.

Definizione del tipo di variabile

I tipi di variabile complessi possono essere definiti utilizzando l'istruzione TYPE...END TYPE. Nell'istruzione DIM, che dichiara una variabile del tipo definito dall'utente, è necessario inserire la parola chiave SHARED, se si desidera che la variabile sia visibile alle procedure del programma, ad esempio:

```
DIM SHARED recordMagazzino AS TipoRecord
```

Se il programma non contiene questa istruzione, i sottoprogrammi e le funzioni devono comprendere l'istruzione:

```
SHARED recordMagazzino AS TipoRecord
```

per poter accedere al contenuto della variabile recordMagazzino.

SOTTOPROGRAMMI

Un *sottoprogramma* è un tipo di procedura; è un blocco di codice indipendente, che esegue una singola attività della programmazione, quindi è utilizzabile senza problemi in molti programmi diversi.

Al contrario della tradizionale subroutine Basic, un sottoprogramma, definito con l'istruzione SUB, ha un solo punto di inizio, in corrispondenza della prima istruzione eseguibile della procedura; può tuttavia avere più di un punto d'uscita dato che, oltre all'istruzione END SUB che contrassegna il normale punto d'uscita, QBasic fornisce l'istruzione EXIT SUB che consente di uscire in anticipo dal sottoprogramma. Le variabili di un sottoprogramma (tranne quelle passate a esso oppure specificate in un'istruzione SHARED) sono *locali.*

Per default, le variabili locali a un sottoprogramma (oppure a una funzione, come viene illustrato di seguito) vengono create ogniqualvolta viene richiamata la procedura e vengono eliminate quando il controllo ritorna alla riga che ha richiamato la procedura; in effetti appartengono soltanto alla procedura che le utilizza e non ne è consentito l'accesso a un altro sottoprogramma, anche se quest'ultimo dovesse contenere variabili con lo stesso nome. Questo consente ai sottoprogrammi di richiamarsi in modo ricorsivo, con ogni iterazione che utilizza la propria serie di variabili. In alcuni casi, tuttavia, si desidera che un sottoprogramma mantenga i valori delle proprie variabili locali tra le chiamate; a questo scopo è sufficiente utilizzare l'istruzione STATIC. Infatti, definendo tutta la procedura come *statica,* non viene eliminata alcuna variabile al termine della sua esecuzione; se invece si rendono statiche soltanto una o due variabili, la parte restante rimane transiente. Si noti tuttavia che esistono due tipi di variabili che sono sempre statiche, ovvero quelle passate al sottoprogramma come argomenti e quelle specificate in un'istruzione SHARED.

Richiamo dei sottoprogrammi

Un sottoprogramma può essere eseguito tramite un'istruzione CALL; ad esempio, l'istruzione di seguito riportata è una chiamata a un sottoprogramma denominato SalvaVGA:

```
CALL SalvaVGA(NomeFigura$)
```

NomeFigura$, la variabile tra parentesi, è un argomento stringa che viene passato al sottoprogramma chiamato. Quando SalvaVGA termina l'esecuzione, qualsiasi modifica apportata a NomeFigura$ nel sottoprogramma viene restituita all'istruzione che l'ha chiamata.

L'istruzione DECLARE può essere utilizzata per specificare il numero e il tipo di variabili utilizzate in una chiamata a un sottoprogramma. Di seguito è riportata un'istruzione DECLARE relativa a SalvaVGA.

```
DECLARE SUB SalvaVGA (NomeFigura AS STRING)
```

Questa istruzione indica al compilatore che quando si richiama SalvaVGA, si richiama in effetti un sottoprogramma (e non una funzione) e che quest'ultimo richiede un solo argomento, che deve essere una stringa.

È bene inserire le istruzioni DECLARE per ogni sottoprogramma o funzione richiamata, anche se, quando un programma viene scritto in ambiente QBasic, l'editor genera queste istruzioni. In effetti, utilizzando DECLARE si rende inutile l'istruzione CALL, ed è possibile richiamare SalvaVGA, semplicemente immettendone il nome:

```
SalvaVGA NomeFigura$
```

Si noti che, se l'istruzione CALL viene omessa, non è necessario utilizzare le parentesi nell'elenco degli argomenti.

SUBROUTINE

Le *subroutine* rappresentano il metodo tradizionale per la memorizzazione del codice che deve essere eseguito ripetutamente nel corso del programma. Per passare il controllo a una sobroutine, è sufficiente utilizzare la seguente sintassi:

```
GOSUB posizione
```

In questa istruzione, posizione rappresenta il numero di riga o l'etichetta di un'istruzione presente nel codice della subroutine. Non è indispensabile che sia la prima istruzione nella subroutine, sebbene inserirla in un punto diverso indichi una scarsa capacità di programmazione. Il programma prosegue dal punto indicato fino a quando individua un'istruzione RETURN, che provoca la ripresa dell'esecuzione con l'istruzione successiva a GOSUB.

Si consiglia di evitare l'utilizzo di quest'ultima istruzione, tenendo conto del fatto che in QBasic le subroutine sono state in gran parte sostituite dai più potenti sottoprogrammi. In ogni modo, esse sono ancora utili in alcuni casi, soprattutto per l'individuazione degli eventi.

FUNZIONI DEFINITE DALL'UTENTE

Le *funzioni* sono semplicemente delle procedure che restituiscono un valore. QBasic non soltanto fornisce molte funzioni già disponibili, ma consente anche di progettarne alcune proprie. I due tipi di funzioni di QBasic sono quelle definite da DEF FN (un residuo di BASICA) e la più potente *procedura function*, che viene definita mediante l'istruzione FUNCTION.

Procedure FUNCTION

Una procedura definita dall'istruzione FUNCTION è simile a un sottoprogramma, tranne per il fatto che restituisce un valore; inoltre deve essere denominata come se fosse una variabile dello stesso tipo del valore restituito. Ad esempio, è possibile dichiarare una funzione che restituisce un intero lungo, come nel seguente caso:

```
DECLARE FUNCTION BinDec& (binario AS STRING)
```

L'istruzione DECLARE specifica il numero e il tipo delle variabili che devono essere inserite nella chiamata a BinDec&. In questo caso non viene utilizzata l'istruzione CALL, ma è sufficiente assegnare il valore restituito dalla funzione a un'altra variabile, come nell'esempio di seguito riportato.

```
decimale& = BinDec&(Numerobinario$)
```

È anche possibile utilizzare una procedura definita da FUNCTION in un'istruzione QBasic, oppure come argomento di un'altra funzione o istruzione, come nell'esempio seguente:

```
PRINT BinDec&("01010011")
OUT altoparlante%, CINT(BinDec&(maschera$))
```

In modo analogo ai sottoprogrammi, le procedure definite da FUNCTION utilizzano argomenti e variabili locali (tranne quelle passate o quelle specificate da un'istruzione SHARED). È necessario rendere statiche le variabili mediante l'istruzione STATIC, se si desidera che i relativi valori vengano mantenuti tra le chiamate. Dato che le variabili sono locali a ogni funzione e che una funzione può essere utilizzata come argomento di un'altra funzione, è possibile richiamare le funzioni in modo ricorsivo.

Funzioni DEF FN

Le funzioni definite dall'istruzione DEF FN rappresentano, per le procedure definite da FUNCTION, ciò che le subroutine sono per i sottoprogrammi. Sono state inserite in QBasic soltanto per garantire la compatibilità con le versioni precedenti di Basic, in cui DEF FN è l'unica funzione definita dall'utente.

Le variabili denominate in un'istruzione DEF FN non sono locali per default, come avviene invece in un'istruzione FUNCTION: per isolare le variabili DEF FN è necessario renderle statiche, in caso contrario la loro visibilità si estende a tutto il programma principale. In pratica, le istruzioni DEF FN non eseguono nulla di diverso dalle istruzioni FUNCTION, ma sono più difficili da utilizzare.

Si consiglia di avvalersi dei sottoprogrammi e delle funzioni invece che delle subroutine e delle funzioni DEF FN, dato che la loro natura modulare le rende semplici da sviluppare e da verificare.

CONCATENAMENTO

Invece di restituire il controllo al DOS, un programma QBasic quando termina è in grado di trasferire il controllo a un altro programma. Questo può essere realizzato in due modi: il primo consiste nell'utilizzare l'istruzione RUN per eseguire un altro programma oppure, se non viene specificato il nome del file, per avviare nuovamente lo stesso programma; il secondo consiste nell'utilizzare l'istruzione CHAIN per avviare un secondo programma. La differenza tra questi due comandi consiste nel fatto che CHAIN consente di passare al nuovo programma le variabili selezionate.

Per passare le informazioni a un altro programma, è necessario utilizzare CHAIN con l'istruzione COMMON. Le variabili specificate da quest'ultima istruzione possono essere trasferite a qualsiasi programma che contenga un'istruzione COMMON corrispondente.

CALL

Si consulti anche

CALL ABSOLUTE, DECLARE, FUNCTION

Scopo

L'istruzione CALL consente di eseguire i sottoprogrammi partendo da un punto qualsiasi del programma.

Sintassi

```
[CALL] nome [[(]arg [,arg...][)]]
```

nome può contenere un massimo di 40 caratteri e deve corrispondere al nome del sottoprogramma da chiamare.

arg è un argomento facoltativo (variabile, costante o espressione) che viene passato al sottoprogramma. Gli argomenti multipli vengono separati da virgole.

CALL è una parola chiave facoltativa nel caso venisse utilizzata l'istruzione DECLARE per definire i tipi di variabili per la chiamata al sottoprogramma. Omettendo CALL, non si devono immettere le parentesi nell'elenco degli argomenti.

Utilizzo

```
CALL ChiudiPorta
```

Esegue la procedura ChiudiPorta, che non assume alcun parametro.

```
CALL Scan(UCASE$(comando$))
```

Esegue il sottoprogramma Scan, che riceve un argomento che è il risultato di una chiamata alla funzione UCASE$ di QBasic.

```
VerMenu, 8, 28, menu%
```

Esegue il sottoprogramma VerMenu, che dispone di una relativa istruzione DECLARE.

Descrizione

I sottoprogrammi sono gruppi di istruzioni indipendenti e quindi non fanno parte del programma principale. Dato che sono entità separate, non è possibile utilizzare le istruzioni GOTO e GOSUB per passare il controllo ai sottoprogrammi; è invece necessario utilizzare l'istruzione CALL.

L'utilizzo dell'istruzione DECLARE per definire in che modo un sottoprogramma viene richiamato elimina la necessità di sfruttare la parola chiave CALL e consente di eseguire il sottoprogramma utilizzando soltanto il relativo nome. Omettendo CALL, è necessario tralasciare le parentesi nell'elenco degli argomenti. In effetti, il sottoprogramma diventa un'estensione del linguaggio QBasic.

Se un argomento è una variabile, deve essere del tipo che il sottoprogramma prevede. Per default, qualsiasi variabile nell'elenco degli argomenti dell'istruzione

CALL viene passata *per riferimento*. Questo significa che QBasic passa al sottoprogramma l'indirizzo delle variabili, non i valori, in modo che qualsiasi modifica apportata alle variabili nel sottoprogramma venga mantenuta, quando il controllo ritorna al programma chiamato. È possibile impedire che ciò avvenga, racchiudendo tra parentesi ogni variabile che non si desidera venga modificata, come nell'istruzione di seguito riportata.

```
QuickSort osservazioni%(), (giorno1%), (giorno6%)
```

Questa istruzione consente a QBasic di valutare la variabile tra parentesi e di copiare il risultato in una variabile temporanea, il cui indirizzo viene quindi passato alla procedura, lasciando invariata la variabile originale. Questa operazione viene spesso denominata passaggio *per valore*.

Avvertenze

Se un sottoprogramma viene richiamato soltanto mediante il proprio nome, QBasic potrebbe erroneamente ritenere il riferimento un'etichetta di riga. Questo succede se il nome del sottoprogramma è la prima istruzione sulla riga e la chiamata non comprende alcun parametro, come nell'esempio di seguito riportato.

```
StampaSchermata: LOCATE 24, 65: PRINT "Premere un tasto >";
```

In questa riga, QBasic potrebbe considerare la chiamata alla procedura StampaSchermata come etichetta di riga, anche se StampaSchermata fosse stato in precedenza dichiarato come sottoprogramma.

In questo caso non viene visualizzato un messaggio di errore; dato che il sottoprogramma non viene mai eseguito, potrebbe essere necessario trascorrere lunghe ore per individuare l'errore prima di comprenderne la ragione. In tali circostanze è necessario inserire sempre la parola chiave CALL, anche se in precedenza è stata utilizzata l'istruzione DECLARE.

Esempio

Il programma di seguito riportato utilizza una procedura per disegnare alcuni riquadri sullo schermo in modalità testo, in modo equivalente all'operazione eseguita dall'istruzione LINE sullo schermo in modalità grafica.

```
DECLARE SUB Riquadro (interruttore%, riga%, colonna%, numrighe%, numcolonne%)

CLS
FOR i% = 1 TO 7
    COLOR 15, i%
    Riquadro 1, i% * 2, i% * 2, 8, 20  'disegna un riquadro con bordo singolo
    Riquadro 2, i% * 2, 40 + i% * 2, 8, 20 'disegna un riquadro con bordo
                                           'doppio
NEXT i%
COLOR 7, 0
END


SUB Riquadro (interruttore%, riga%, colonna%, numrighe%, numcolonne%) STATIC
    IF numcolonne% > 81 - colonna% THEN numcolonne% = 81 - colonna%
    IF numrighe% > 24 - riga% THEN numrighe% = 24 - riga%
```

```
    IF interruttore% = 1 THEN        'caratteri per il riquadro a bordo singolo
        sup$ = CHR$(218) + STRING$(numcolonne% - 2, CHR$(196)) + CHR$(191)
        centr$ = CHR$(179) + STRING$(numcolonne% - 2, " ") + CHR$(179)
        inf$ = CHR$(192) + STRING$(numcolonne% - 2, CHR$(196)) + CHR$(217)

    ELSE                              'caratteri per il riquadro a bordo doppio
        sup$ = CHR$(201) + STRING$(numcolonne% - 2, CHR$(205)) + CHR$(187)
        centr$ = CHR$(186) + STRING$(numcolonne% - 2, " ") + CHR$(186)
        inf$ = CHR$(200) + STRING$(numcolonne% - 2, CHR$(205)) + CHR$(188)
    END IF
    LOCATE riga%, colonna%, 0: PRINT sup$; 'stampa la parte superiore del
                                           'quadrato
    FOR i% = 1 TO numrighe% - 2
        LOCATE riga% + i%, colonna%: PRINT centr$;  'stampa la parte centrale
                                                    'del quadrato
    NEXT i%
    LOCATE riga% + i%, colonna%: PRINT inf$;        'stampa la parte inferiore
                                                    'del quadrato
END SUB
```

CHAIN

Si consulti anche

CALL, COMMON, SHELL

Scopo

Quando il compito svolto da un programma è terminato, l'istruzione CHAIN consente di passare il controllo a un altro programma. Questa istruzione deve essere utilizzata quando un certo compito viene suddiviso tra diversi programmi.

Sintassi

```
CHAIN nomefile
```

nomefile è il nome del programma a cui ci si concatena. In modo analogo ai nomi di file del DOS, non può avere una lunghezza superiore agli otto caratteri; è possibile includere la lettera dell'unità, il percorso della directory e l'estensione del nome del file. Se il nome viene immesso letteralmente, è necessario racchiuderlo tra doppi apici; in caso contrario, è possibile utilizzare qualsiasi variabile stringa corretta o un'espressione che possa contenere il nome del file.

Utilizzo

```
CHAIN "A:\BIN\PROGRAMMA2"
```

Trasferisce il controllo al programma PROGRAMMA2 presente nell'unità A: all'interno della directory BIN.

```
CHAIN unita$ + percorso$ + programma$(scelta%)
```

Trasferisce il controllo al programma specificato dalla concatenazione delle variabili unita$, percorso$ e programma$(scelta%).

Descrizione

Dato che QBasic è un linguaggio modulare, è possibile inserire i sottoprogrammi e le funzioni nei programmi. Per quanto riguarda le applicazioni di ampie dimensioni, la scelta migliore potrebbe essere quella di suddividere il compito tra diversi programmi differenti, ognuno dedicato a un aspetto particolare del lavoro. L'istruzione CHAIN consente di trasferire il controllo da un programma a un altro.

È anche possibile passare le variabili che vengono specificare in un'istruzione COMMON nel programma "concatenante" al programma a cui deve essere "concatenato", a patto che entrambi i programmi presentino istruzioni COMMON corrispondenti.

CHAIN gestisce i programmi che vengono eseguiti in ambiente QBasic in modo diverso rispetto alla modalità utilizzata per gestire i programmi eseguiti dalla riga di comando del DOS: quando i programmi vengono concatenati, il programma di destinazione deve essere un file di QBasic. A meno che non venga specificato diversamente, QBasic assume l'estensione BAS.

Avvertenze

Si noti che quando si esegue il concatenamento tra i programmi di QBasic, qualsiasi file lasciato aperto dal programma concatenante rimane aperto quando il programma concatenato viene avviato. Dato che QBasic consente di aprire lo stesso file più di una volta, potrebbe succedere talvolta di riaprire casualmente diversi file, fino a quando non viene visualizzato il messaggio di errore: "Troppi file".

Esempio

Per semplificare la programmazione, le applicazioni di ampie dimensioni vengono spesso suddivise in diversi programmi più piccoli od *overlay*. Il programma di seguito riportato presenta un menu di alto livello per ogni applicazione ed è sufficientemente flessibile da poter essere utilizzato per diverse applicazioni. Dopo che l'utente ha scelto un'opzione del menu, il programma utilizza l'istruzione CHAIN per eseguire il programma corrispondente a quella specifica opzione.

```
'$DYNAMIC

CLS : RESTORE
READ opzioni%                     'ottiene il numero di opzioni nel menu
DIM menu(opzioni%) AS STRING      'dichiara un array di stringhe
READ menu(0)                 'ottiene la stringa contenente le possibili scelte

FOR i% = 1 TO opzioni%
    READ menu(i%)
    IF i% = 1 THEN colonna% = 40 - (LEN(menu(i%)) \ 2)
    LOCATE 8 + i%, colonna%: PRINT menu(i%);       'stampa le opzioni del menu
NEXT i%

LOCATE 9 + i%, colonna%: PRINT "Premi <Esc> per uscire";
LOCATE 12 + i%, colonna% + 3: PRINT "< La tua scelta";

DO
    LOCATE 12 + i%, colonna%, 1
    scelta$ = INPUT$(1): IF scelta$ = CHR$(27) THEN EXIT DO
    scelta$ = UCASE$(scelta$)
```

```
    selezione% = INSTR(menu(0), scelta$)         'trova l'indice della scelta
    IF selezione% THEN
        PRINT scelta$;
        CHAIN "PROG" + LTRIM$(RTRIM$(STR$(selezione%))) 'trasferisce il
                                                        'controllo a un
                                                        'programma
    END IF
    BEEP
LOOP WHILE 1
END

DATA    6, "CAMCOS"
DATA    "C. Crea un file di dati     ","A. Aggiungi un nuovo record"
DATA    "M. Modifica un record       ","C. Cancella un record      "
DATA    "O. Ordina il file           ","S. Stampa i rapporti       "
```

COMMON

Si consulti anche

CHAIN, $DINANIC, FRE

Scopo

L'istruzione COMMON determina la visibilità delle variabili presenti nell'elenco indicato. Questa istruzione estende la visibilità delle variabili elencate ai programmi concatenati che presentano un'istruzione COMMON corrispondente.

Sintassi

```
COMMON SHARED var [()] [AS tipo] [,var [()] [AS tipo]...]
```

SHARED è un'opzione che consente di estendere la visibilità delle variabili presenti nell'elenco, rendendole disponibili ai sottoprogrammi e alle funzioni nel programma e ai programmi concatenati. Dato che le variabili dichiarate in un'istruzione COMMON con la parola chiave SHARED possono essere esaminate o modificate da tutti i sottoprogrammi e le funzioni richiamate nel programma principale, non è necessario dichiarare ognuna di esse come condivisa all'interno dei sottoprogrammi e delle funzioni che le utilizzano.

var rappresenta un qualsiasi nome valido di variabile QBasic e può comprendere un carattere di dichiarazione del tipo (ad esempio % per gli interi). Tuttavia, l'utilizzo della clausola AS risulta più flessibile dato che consente di inserire i nomi dei tipi definiti dall'utente oltre ai tipi standard INTEGER, LONG, SINGLE, DOUBLE e STRING. Le variabili array vengono identificate dalle parentesi.

Utilizzo

```
COMMON saldo, Cognome AS STRING
```

Dichiara un variabile a precisione semplice e una stringa a lunghezza variabile, rendendole disponibili per i programmi concatenati.

```
COMMON SHARED eta() AS INTEGER
```

Dichiara un array intero dinamico e lo rende disponibile per tutti i sottoprogrammi, le funzioni e i programmi concatenati.

```
DIM AnniLuce(1 TO 50) AS INTEGER
COMMON AnniLuce()
```

Dichiara un array intero lungo statico composto da 50 elementi e lo rende disponibile per i programmi concatenati

Descrizione

Le istruzioni COMMON assegnano la memoria per i dati globali durante l'inizializzazione del programma; devono essere inserite nella parte superiore del programma principale, prima di qualsiasi istruzione eseguibile. Le sole istruzioni che possono precedere COMMON sono i *metacomandi* ($DYNAMIC e $STATIC), le dichiarazioni delle costanti e delle procedure (CONST, DATA e DECLARE), le dichiarazioni del tipo dei dati che utilizzano DEFtipo, oppure le istruzioni TYPE...END TYPE, SHARED, STATIC, OPTION BASE e DIM. Gli array statici denominati nell'elenco delle variabili dell'istruzione COMMON devono essere dichiarati con istruzioni DIM prima dell'istruzione COMMON stessa, altrimenti sono considerati dinamici ed è quindi possibile utilizzare le istruzioni DIM o REDIM per dimensionarli o ridimensionarli.

Corrispondenti istruzioni COMMON devono essere inserite in tutti i programmi che devono accedere ai dati globali. Si noti tuttavia che QBasic non mantiene i nomi delle variabili tra i programmi, ne consegue che è possibile utilizzare nomi diversi per fare riferimento alle stesse variabili in programmi differenti. In ogni caso, è necessario mantenere il tipo e la posizione delle variabili nell'istruzione COMMON, come riportato nella Figura 4.1.

```
'Programma A
COMMON uno%, due#,  tre$
          |      |      |
          |      |      |
'Programma B     |      |
COMMON un%, deux#, trois$
```

Figura 4.1 Corrispondenza delle variabili COMMON

Se un array common viene dichiarato nel programma principale come statico, è indispensabile dichiararlo statico anche in tutti gli altri programmi che lo utilizzano. In modo analogo, un array dinamico deve essere dinamico in tutti i programmi di cui fa parte, dato che QBasic memorizza gli array statici e dinamici in segmenti diversi della memoria.

Il numero delle variabili common in un programma concatenato non deve necessariamente essere identico a quello del programma concatenante. Ad esempio, se il programma A dichiara quattro variabili common e il programma B utilizza soltanto le prime due, in quest'ultimo programma è necessario dichiarare soltanto due variabili, come illustrato nel seguente esempio.

```
'Programma A
COMMON uno AS INTEGER, due AS DOUBLE, tre AS STRING, quattro AS LONG
'Programma B
COMMON uno AS INTEGER, due AS DOUBLE
```

Tuttavia, se il programma B deve accedere alle variabili uno e quattro, è necessario dichiarare tutte e quattro le variabili come common nel programma B, anche se due e tre non vengono utilizzate.

```
'Programma A
COMMON uno AS INTEGER, due AS DOUBLE, tre AS STRING, quattro AS LONG
'Programma B
COMMON uno AS INTEGER, x#, y$, quattro AS DOUBLE
```

Commenti

QBasic non pone alcun limite per quanto riguarda il numero di variabili che possono essere dichiarate come common; tuttavia, dato che queste variabili sono allocate nel DGROUP (il segmento dati di default del programma), è necessario lasciare uno spazio sufficiente di questi 32 KB per le altre variabili utilizzate dal programma. Si utilizzi l'istruzione FRE("") per determinare il numero di byte liberi rimasti in DGROUP.

Gli array numerici dinamici rappresentano un'eccezione a questo limite, dato che utilizzano soltanto lo spazio sufficiente in DGROUP per memorizzare i nomi degli array; il loro contenuto viene memorizzato nel *far heap* (un'area di memoria contenente gli array numerici e gli array di stringhe a lunghezza fissa). L'istruzione FRE(-1) restituisce il numero di byte disponibili nel far heap per gli array numerici dinamici.

Errori

Se una variabile common è di un tipo diverso rispetto a quello della variabile che occupa la stessa posizione nell'istruzione COMMON di un programma concatenato, QBasic visualizza al momento dell'esecuzione il messaggio di errore: "Mancata corrispondenza dei tipi".

Suggerimenti

Si consiglia di non assegnare lo stesso nome alle variabili dichiarate in istruzioni COMMON diverse all'interno dello stesso programma. Non sempre questa operazione produce il previsto messaggio di errore: "Duplice definizione" al momento dell'esecuzione, e può dare come risultato il passaggio di dati errati tra i programmi. Questo è un tipo di errore che potrebbe essere molto difficile individuare.

È inoltre necessario tenere presente che le variabili common non sono disponibili ai sottoprogrammi e alle funzioni nel programma in cui vengono dichiarate, a meno che non sia stata specificata l'opzione SHARED.

Esempio

I due programmi di seguito riportati illustrano l'utilizzo dell'istruzione COMMON. Il primo richiede l'immissione del nome dell'utente e quindi si concatena al secondo programma, che richiama il sottoprogramma Immisione, che è in grado di accettare qualsiasi tasto valido immesso dall'utente.

```
DECLARE SUB Immissione (buffer$)

COMMON SHARED termine AS INTEGER, maxLungh AS INTEGER, validoTasto AS STRING

CONST FALSO = 0, VERO = NOT FALSO          'costanti usate da questo programma

Immissione tuonome$

LOCATE 15, 1
IF termine THEN
    PLAY "T255 L24 01 A A A 02 B B B 03 C C C 04 D D D"
    COLOR 12, 0: PRINT "Hai premuto Esc"
ELSE
    PLAY "T240 03 L8 G F L4 E L8 E E D E L3 F L4 E"
    PLAY "L8 E L4 D L8 D D C D L3 E L4 C L8 D L4 E"
    PLAY "L8 E E D E L4 F L8 G L4 A L8 A L4 G L8 F L4 E L8 D L4 C"
    COLOR 11, 0: PRINT "Ciao, "; tuonome$
END IF
LOCATE , , 1
END

SUB Immissione (buffer$) STATIC
   backSpace$ = CHR$(8): cr$ = CHR$(13): esc$ = CHR$(27)       'tasti speciali
   curSinistra$ = CHR$(29): cursore$ = CHR$(95): curFlag% = FALSO  'variabili
                                                                   'per il
                                                                   'cursore
   puntatore% = 0: buffer$ = "": fatto% = FALSO: termine% = FALSO
   segno& = 0: inizio% = POS(0)

   PRINT SPACE$(maxLungh);
   LOCATE , inizio%, 0
   DO
       DO                                  'visualizza un cursore lampeggainte
           IF ABS(TIMER - segno&) > .3 THEN
               segno& = TIMER
               IF curFlag% THEN
                   PRINT " "; curSinistra$;        'cancella il cursore
                   curFlag% = FALSO
               ELSE
                   PRINT cursore$; curSinistra$;   'disegna il cursore
                   curFlag% = VERO
               END IF
           END IF
           tasto$ = INKEY$          'ottiene un carattere dall'utente
       LOOP WHILE tasto$ = ""
       SELECT CASE tasto$
           CASE esc$               'se il carattere è Esc
               termine = VERO
               fatto% = VERO
           CASE cr$                'se il carattere è Invio
               fatto% = VERO
           CASE backSpace$         'se il carattere è Backspace
               IF puntatore% > 0 THEN
                   PRINT " "; curSinistra$; curSinistra$;
                   puntatore% = puntatore% - 1
                   buffer$ = LEFT$(buffer$, puntatore%)
               END IF
           CASE ELSE               'controlla la validità del carattere
               IF INSTR(validoTasto, tasto$) AND puntatore% < maxLungh THEN
                   PRINT tasto$;
                   puntatore% = puntatore% + 1
                   buffer$ = buffer$ + tasto$
               END IF
       END SELECT
   LOOP UNTIL fatto%
```

```
        PRINT " "; curSinistra$;
END SUB


COMMON SHARED termine AS INTEGER, maxLungh AS INTEGER, validoTasto AS STRING

CONST FALSO = 0, VERO = NOT FALSO          'costanti usate da questo programma


minusc$ = "abcdefghijklmnopqrstuvwxyz"
maiusc$ = UCASE$(minusc$)
num$ = "0123456789"
punt$ = " .,;:'"

COLOR 14, 0: CLS
LOCATE 12, 1: PRINT "Inserisci il tuo nome > "; 'richiesta del nome
COLOR 0, 7
maxLungh = 40
validoTasto = maiusc$ + minusc$ + punt$   'stringa contenente i tasti validi

CHAIN "4-3"
```

DECLARE

Si consulti anche

CALL, FUNCTION, SUB

Scopo

L'istruzione DECLARE definisce la modalità in base alla quale vengono richiamate le procedure FUNCTION o SUB; specifica il nome di una procedura e definisce il numero e il tipo delle variabili che vengono passate a essa come argomenti. L'istruzione DECLARE elimina la necessità di utilizzare la parola chiave CALL.

Sintassi

```
DECLARE FUNCTION Nome [([elencoparametri])]
DECLARE SUB Nome [([elencoparametri])]
```

Nome è il nome della procedura da richiamare; può comprendere un massimo di 40 caratteri e deve corrispondere al nome della procedura a cui fa riferimento.

elencoparametri è l'elenco delle variabili che sono i parametri della procedura. I tipi di variabili devono essere specificati nell'elenco.

Utilizzo

```
DECLARE SUB Verifica (stringa$, ok%)
```

Definisce un sottoprogramma denominato Verifica e specifica il tipo di argomenti passati.

```
DECLARE FUNCTION Centro& (testo AS STRING, LarghezzaMassima AS INTEGER)
```

Dichiara Centro& come una funzione definita dall'utente che assume come argomenti una stringa e un intero.

Descrizione

Le istruzioni DECLARE possono essere presenti soltanto nel codice del programma principale e vengono generalmente elencate nella parte superiore, prima di qualsiasi istruzione eseguibile. Queste istruzioni definiscono i nomi delle procedure che vengono richiamate nel programma e specificano inoltre il numero e il tipo di variabili da passare coma argomenti. Le istruzioni DECLARE vengono spesso denominate *prototipi* delle procedure a cui fanno riferimento.

Se la procedura richiamata restituisce un valore, essa deve essere definita con l'istruzione FUNCTION, nel qual caso il nome della procedura viene interpretato come una variabile. Per specificare il tipo di valore restituito da un'istruzione FUNCTION, è sufficiente utilizzare un carattere di dichiarazione del tipo (%, &, !, # o $). Se invece la procedura non restituisce un valore, occorre dichiararla con la parola chiave SUB.

I nomi delle variabili nell'elenco dei parametri dell'istruzione DECLARE non vengono passati alle procedure esterne e non è necessario che corrispondano a quelli utilizzati nella procedura chiamata. Tuttavia è importante che i tipi di variabile utilizzati nella chiamata alla procedura siano identici a quelli nelle posizioni corrispondenti dell'elenco dei parametri dell'istruzione DECLARE. I tipi di variabile possono essere specificati inserendo un carattere di dichiarazione del tipo, utilizzando una clausola AS con INTEGER, LONG, SINGLE, DOUBLE, STRING oppure indicando il nome di un tipo definito dall'utente per dichiarare la variabile di un tipo particolare. Indipendentemente dal metodo utilizzato, le stringhe a lunghezza fissa non devono essere utilizzate come argomenti, dato che soltanto quelle a lunghezza variabile possono essere passate alle funzioni e ai sottoprogrammi.

E inoltre possibile passare come parametri degli array (ad esempio per ordinarli). In questi casi non è necessario specificare il numero delle dimensioni nell'elenco dei parametri dell'istruzione DECLARE; se tale numero viene invece inserito, l'editor di QBasic lo accetta, ma poi lo cancella.

Errori

Se viene richiamata una procedura che presenta una variabile di un tipo diverso da quella presente nella corrispondente posizione nell'elenco dei parametri dell'istruzione DECLARE, QBasic visualizza il messaggio di errore: "Non corrispondono i tipi dei parametri". Se il numero di parametri è maggiore o minore rispetto a quello dei parametri elencati nell'istruzione DECLARE, QBasic visualizza il messaggio di errore: "Numero degli argomenti non corrispondente".

Esempio

L'esempio di seguito riportato comprende due procedure, ovvero BinDec&, una funzione, e DecBin, un sottoprogramma, che consentono di convertire i numeri tra il formato binario e quello decimale. Le istruzioni DECLARE indicano al compilatore che tipo di procedure sono BinDec& e DecBin e quindi specificano il numero e il tipo di argomenti previsti.

```
'dichiara le procedure
DECLARE FUNCTION BinDec& (binario AS STRING)
DECLARE SUB DecBin (decimale AS LONG, bin AS STRING)
```

```
decimale& = 123456                       'un valore decimale
DecBin decimale&, bin$                   'conversione da decimale a binario
PRINT "Binario = "; bin$                 'visualizza il risultato
PRINT "Decimale ="; BinDec&(bin$)        'conversione da binario a decimale
END

'converte una stringa di numeri binari nei loro equivalenti decimali

FUNCTION BinDec& (binario$) STATIC
    decimale& = 0: potenza% = 0
    binario$ = UCASE$(binario$)
    FOR i% = LEN(binario$) TO 1 STEP -1    'conversione da binario a decimale
        numero% = ASC(MID$(binario$, i%, 1)) - 48
        IF numero% < 0 OR numero% > 1 THEN decimale& = 0: EXIT FOR
        decimale& = decimale& + numero% * 2 ^ (potenza%)
        potenza% = potenza% + 1
    NEXT i%
    BinDec& = decimale&
END FUNCTION


'converte un valore decimale in una stringa equivalente di numeri binari
SUB DecBin (decimale&, bin$) STATIC
    bin$ = ""
    h$ = HEX$(decimale&)                'conversione da decimale a esadecimale
    FOR i% = 1 TO LEN(h$)
        numero% = INSTR("0123456789ABCEDF", MID$(h$, i%, 1)) - 1
        IF numero% < 0 THEN bin$ = "": EXIT FOR
        j% = 8: k% = 4
        DO                                  'conversione da esadecimale a binario
            bin$ = bin$ + RIGHT$(STR$((numero% \ j%) MOD 2), 1)
            j% = j% - (j% \ 2): k% = k% - 1
            IF k% = 0 THEN EXIT DO
        LOOP WHILE j%
    NEXT i%
END SUB
```

DEF FN

Si consulti anche

FUNCTION, STATIC, SUB

Scopo

DEF FN consente di definire una funzione. QBasic fornisce questo comando in modo che i programmi scritti con le versioni precedenti di Basic che supportano soltanto l'istruzione DEF FN per le funzioni definite dall'utente possano essere facilmente trasportati in QBasic. L'istruzione DEF FN è stata ampliata grazie all'eliminazione delle restrizioni che la limitavano a una singola riga di programma.

Sintassi

Sintassi 1:

```
DEF FNNome [(arg[AS tipo] [,arg[AS tipo]...)] = espressione
```

Sintassi 2:

```
DEF FNNome [(arg[AS tipo] [,arg[AS tipo...])]
   [STATIC var [,var...]]
   .
   .
   .
   [EXIT DEF]
   .
   .
   .
   FNNome = espressione
   .
   .
   .
END DEF
```

Nome rappresenta un qualsiasi nome di variabile corretto di QBasic e deve essere sempre preceduto dal prefisso FN, che indica che è una funzione. Dato che QBasic considera Nome come una variabile, è necessario inserire un carattere di dichiarazione del tipo; in caso contrario, viene restituito un valore a precisione semplice (default) oppure il tipo definito nell'istruzione DEFtipo.

arg è il nome di un argomento facoltativo che viene passato alla funzione. È necessario inserire un carattere di dichiarazione del tipo, oppure utilizzare la parola chiave AS con INTEGER, LONG, SINGLE, DOUBLE o STRING per definire un argomento come un tipo particolare. Gli array, le stringhe a lunghezza fissa e i tipi di variabile definiti dall'utente non possono essere passati come argomenti a una funzione DEF FN.

Le funzioni DEF FN a righe multiple dispongono di altre due caratteristiche: possono utilizzare variabili diverse da quelle che vengono passate come argomenti; tuttavia queste variabili sono visibili (e possono essere modificate) alle istruzioni presenti nella parte restante del programma. Per risolvere eventuali problemi, è sufficiente utilizzare la parola chiave STATIC per dichiarare esplicitamente che le variabili sono locali alla funzione.

Utilizzando l'istruzione EXIT DEF, è possibile far terminare una funzione a righe multiple, prima che venga individuato un END DEF. Se a FNNome non è stato assegnato alcun valore quando viene eseguita l'istruzione EXIT DEF, la funzione restituisce 0 per i tipi numerici e una stringa nulla ("") per i tipi stringa.

Utilizzo

```
DEF FNBitTest (numero%, bit%) = -SGN(numero% AND 2 ^ bit%)
```

Questa istruzione DEF FN su riga singola verifica il bit specificato dell'intero numero%, restituendo -1 (vero) se bit% è impostato e 0 (falso) in caso contrario.

```
DEF FNMassimo! (a!, b!, c!)
   STATIC max!
   IF a! > b! THEN
      max! = a!
   ELSEIF c! > b! THEN
      max! = c!
   ELSE max! = b!
   END IF
   FNMassimo! = max!
END DEF
```

Questa istruzione DEF FN a righe multiple individua il valore massimo tra i tre valori inseriti. Si noti che la variabile max! viene dichiarata in un'istruzione STATIC per renderla locale alla funzione.

Descrizione

DEF FN viene tradizionalmente utilizzata per aggiungere funzioni specifiche alla libreria standard di funzioni e di istruzioni utilizzata in Basic. Questa istruzione consente di assegnare qualsiasi espressione a una variabile il cui nome inizi con la parola chiave FN. Quindi, QBasic valuta l'espressione ogniqualvolta il programma fa riferimento a quella funzione DEF FN. L'esempio di seguito riportato illustra questa procedura.

```
DEF FNImpostaBit(numero%, bit%) = numero% OR 2 ^ bit%
IF FNImpostaBit(attributo%, 7) THEN Lampeggia(carattere%)
```

Le funzioni DEF FN sono simili alle subroutine ma, oltre a essere più compatte ed efficaci, restituiscono un valore e possono essere utilizzate come parte di un'altra espressione e persino richiamate dall'interno di altre funzioni o istruzioni, come nel seguente esempio:

```
DEF FNImpostaBit(numero%, bit%) = numero% AND (32767 - 2 ^ bit%)
PRINT CHR$(FNImpostaBit(ASC("a"), 5))
```

In questo caso, la funzione ASC (inserita all'interno di DEF FN) fornisce a FNImpostaBit il valore 97 (ovvero il valore ASCII di "a"). Impostando il bit 5 di questo numero a 0, esso viene ridotto di 32 (a 65) e la funzione CHR$ converte il valore 65 in A.

Istruzioni a righe multiple

QBasic consente di utilizzare il necessario numero di righe prima di assegnare un valore alla funzione. In questo caso, tuttavia, è indispensabile utilizzare anche l'istruzione END DEF per contrassegnare la fine della funzione e restituire il controllo alla riga che l'ha richiamata. Se viene utilizzata l'istruzione EXIT DEF, la funzione termina in anticipo e il controllo viene restituito senza che sia stata eseguita un'assegnazione alla funzione.

L'istruzione FUNCTION consente di creare funzioni definite dall'utente separate rispetto al programma principale. Dato che queste funzioni possono richiamarsi in modo ricorsivo, rappresentano sicuramente il metodo migliore per creare nuove funzioni. Le funzioni DEF FN non possono essere utilizzate in modo ricorsivo.

Errori

QBasic cerca di convertire il valore restituito da un'istruzione DEF FN con la precisione specificata dal tipo del nome di funzione. Se ciò non è possibile perché, ad esempio, l'espressione restituisce un valore numerico e la funzione viene definita come stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Le dichiarazioni DEF FN devono essere inserite prima di qualsiasi istruzione che vi faccia riferimento. Se una funzione viene richiamata prima di essere dichiarata, QBasic visualizza il messaggio di errore: "Funzione non definita".

Esempio

Il programma di seguito riportato utilizza l'istruzione DEF FN per definire una funzione che restituisce una stringa contenente il giorno della settimana di una data qualsiasi compresa nell'intervallo da 1901 a 2099. Si noti che le variabili numGiorno!, numAnno! e giorni$ vengono dichiarate in un'istruzione STATIC per renderle locali alla funzione; per utilizzare quest'ultima nei propri programmi, è sufficiente specificare i parametri nei seguenti intervalli:

```
m% = mese(1 - 12), g% = giorno(1 -31), a% = anno(1901 - 2099)

DEF FNgiornoSettimana$ (g%, m%, a%)
     STATIC numGiorno!, numAnno!, giorni$

     numAnno! = a%
     IF numAnno! < 1901 OR numAnno! > 2099 THEN EXIT DEF

     giorni$ = "Venerdì Sabato   Domenica Lunedì   Martedì  Mercoledì Giovedì"
     num! = VAL(MID$("000303060811131619212426", (m% - 1) * 2 + 1, 2))
     numGiorno! = numAnno! * 365 + INT((a% - 1) \ 4) + (m% - 1) * 28
     numGiorno! = numGiorno! + num! - ((m% > 2) AND (a% MOD 4 = 0)) + g%

     FNgiornoSettimana$ = MID$(giorni$, (numGiorno! - INT(numGiorno! \ 7) * 7)
     * 9 + 1, 9)
END DEF

CLS
PRINT "Il 1 Gennaio 2000 cade di "; FNgiornoSettimana$(1, 1, 2000)
```

FUNCTION

Si consulti anche

DECLARE, DEF FN, EXIT, SHARED, STATIC, SUB

Scopo

QBasic dispone di diverse funzioni predefinite che eseguono calcoli matematici, oppure restituiscono informazioni relative a una variabile. Se nessuna funzione è in grado di eseguire il compito di programmazione desiderato è possibile utilizzare l'istruzione FUNCTION, che consente di definirne una personalizzata.

Sintassi

```
FUNCTION Nome [(arg[AS tipo][, arg[AS tipo]]...)] [STATIC]
   [SHARED var [, var...]]
   [STATIC var [, var...]]
   .
   .
   .
[EXIT FUNCTION]
   .
   .
   .
   Nome = espressione
   .
   .
   .
END FUNCTION
```

Nome è il nome assegnato alla funzione e deve essere un nome di variabile valido e comprendere un carattere di dichiarazione del tipo, se necessario.

arg è il nome di una variabile passata alla funzione come argomento e deve comprendere un carattere di dichiarazione del tipo, essere dichiarato mediante la clausola AS con INTEGER, LONG, SINGLE, DOUBLE, STRING, oppure può essere un tipo definito dall'utente. Gli array passati come argomenti devono comprendere le parentesi dopo il loro nome, sebbene non sia necessario specificare il numero delle dimensioni.

L'istruzione SHARED specifica le variabili presenti nel programma principale che devono essere utilizzate dalle funzioni, ma che non vengono specificamente passate come argomenti.

L'istruzione STATIC specifica le variabili locali i cui valori vengono mantenuti tra le chiamate alla funzione. Tralasciando questa istruzione, nuove variabili locali vengono create ogniqualvolta la funzione viene richiamata.

L'istruzione EXIT FUNCTION restituisce il controllo al programma chiamante, prima che venga raggiunto il normale punto di uscita (l'istruzione END FUNCTION).

Utilizzo

```
FUNCTION conSegno% (senzaSegno&) STATIC
   IF senzaSegno& > 32767 THEN
      conSegno% = senzaSegno& - 65536
   ELSE
      conSegno% = senzaSegno&
   END IF
END FUNCTION
```

Questa funzione converte un intero lungo di 4 byte senza segno in un valore intero equivalente di 2 byte con segno; può essere utile per accedere agli indirizzi di memoria presenti sopra la posizione 32767.

Descrizione

Una funzione è un insieme separato di istruzioni che restituisce un valore. Il valore restituito da una funzione definita da un'istruzione FUNCTION viene generalmente assegnato a un'altra variabile, tuttavia è anche possibile applicare il valore direttamente a un'altra funzione o istruzione di QBasic. È persino possibile utilizzare una funzione come argomento per un'altra funzione o sottoprogramma definiti dall'utente. L'istruzione di seguito riportata rappresenta un'applicazione corretta della funzione conSegno%.

```
PRINT PEEK(conSegno%(65000))
```

Dato che una funzione restituisce sempre un valore, QBasic gestisce il nome della funzione come variabile. Per questo motivo il nome deve comprendere il carattere corretto di dichiarazione del tipo in base al valore che restituisce; se questo carattere non viene inserito, QBasic presuppone che il valore restituito sia a precisione semplice o del tipo definito dall'istruzione DEFtipo.

Generalmente le funzioni assumono una o più variabili come argomenti. I tipi di queste variabili possono essere definiti utilizzando il carattere di dichiarazione del tipo corretto, la clausola AS con INTEGER, LONG, SINGLE, DOUBLE, STRING, oppure il

nome di un tipo definito dall'utente. Gli array passati alla funzione come argomenti devono comprendere le parentesi dopo il loro nome, sebbene non sia necessario elencare il numero delle dimensioni.

All'interno di una funzione definita dall'utente è possibile specificare un numero qualsiasi di variabili; per default esse sono presenti soltanto durante l'esecuzione della funzione e il relativo contenuto viene eliminato quando il controllo ritorna alla riga che ha richiamato la funzione. Per mantenere il contenuto delle variabili locali tra le chiamate, è sufficiente utilizzare la parola chiave STATIC. Questa operazione permette di velocizzare in modo considerevole l'esecuzione del programma, dato che, dopo la prima chiamata, non è più necessario che QBasic crei le variabili locali ogni volta che la funzione viene richiamata. È anche possibile sfruttare l'istruzione STATIC al posto dell'omonima parola chiave per mantenere tra le chiamate soltanto alcuni valori delle variabili locali. Queste ultime sono disponibili soltanto per la funzione in cui vengono utilizzate; questo significa che alle altre funzioni o istruzioni presenti nel codice da cui viene richiamata la funzione non è possibile utilizzarle. In modo analogo, le variabili definite nel programma principale sono generalmente invisibili alle funzioni definite dall'utente, a meno che non vengano specificatamente passate come argomenti. Dato che l'istruzione SHARED consente di ignorare questa condizione di default, essa deve essere utilizzata per specificare le variabili nel programma principale a cui la funzione deve accedere. È tuttavia necessario ricordare che qualsiasi modifica apportata alle variabili all'interno di una funzione viene restituita al programma principale quando quest'ultimo ottiene nuovamente il controllo. Le variabili condivise devono essere dichiarate all'inizio della funzione, prima di qualsiasi istruzione eseguibile. Se una variabile è già stata definita come condivisa all'interno del programma principale, non è necessario definirla nuovamente nella funzione per poterla utilizzare.

Errori

Sebbene QBasic consideri il nome di funzione come una variabile, non è possibile definire il tipo del valore restituito utilizzando la clausola AS. Ad esempio, l'istruzione di seguito riportata provoca la visualizzazione del messaggio di errore: "Era atteso: (o fine istruzione".

```
FUNCTION HiByte AS INTEGER (numero AS INTEGER) STATIC
```

È tuttavia possibile utilizzare l'istruzione DEFtipo come metodo alternativo per definire il tipo di funzione, ad esempio:

```
DEFINT H
FUNCTION HiByte (numero AS INTEGER) STATIC
```

consente di definire correttamente HiByte come funzione intera.

La maggior parte degli utenti preferisce tuttavia utilizzare il metodo standard per assegnare un tipo di variabile alla funzione, ovvero:

```
FUNCTION HiByte% (numero%) STATIC
```

Avvertenze

Ogniqualvolta la parola chiave STATIC viene utilizzata in una FUNCTION, è necessario tenere presente che le variabili locali mantengono i valori a loro assegnati

durante le chiamate precedenti alla funzione. A meno che queste variabili non vengano inizializzate ogni volta che la funzione viene richiamata, questa potrebbe restituire un valore sbagliato. Questo non è soltanto uno degli errori di programmazione più comuni, ma è anche uno dei più difficili da individuare.

Esempio

La conversione delle frazioni in numeri decimali è un'operazione semplice, dato che è sufficiente dividere il numeratore per il denominatore; non è tuttavia facile convertire i decimali in frazioni. La funzione QBasic di seguito riportata esegue questa operazione, trasformando il valore del denominatore in variabile, in modo che le frazioni possano essere calcolate con diversi gradi di precisione.

```
DECLARE FUNCTION ConDecFraz$ (numero!)
'converte un numero con parte decimale in un numero intero e in una frazione

FUNCTION ConDecFraz$ (numero!) STATIC
    SHARED denominatore%                        'valore del denominatore
    fraz$ = ""
    parteIntera& = INT(numero!)                 'ottiene la parte intera
    decimale! = numero! - parteIntera&          'ottiene la parte decimale
    IF decimale! > 0 THEN
        numeratore% = INT(denominatore% * decimale!)  'ottiene il numeratore
                                                       'della frazione
        IF numeratore% = 0 THEN numeratore% = 1
        DO WHILE numeratore% MOD 2 = 2 = 0 AND denominatore% MOD 2 = 0
            numeratore% = numeratore% \ 2
            denominatore% = denominatore% \ 2
        LOOP        'cicla mentre numeratore e denominatore sono entrambi pari
        fraz$ = STR$(numeratore%) + "/" + LTRIM$(STR$(denominatore%))
    END IF
    ConDecFraz$ = LTRIM$(RTRIM$(STR$(parteIntera&))) + fraz$
END FUNCTION
```

GOSUB...RETURN

Si consulti anche

CLEAR, GOTO, ON...GOSUB, SUB

Scopo

L'istruzione GOSUB consente di eseguire un salto incondizionato a una subroutine in un punto qualsiasi dello stesso modulo; RETURN contrassegna un'uscita dalla subroutine. Questa istruzione permette di riprendere l'esecuzione in corrispondenza dell'istruzione successiva a quella che ha richiamato la subroutine, oppure di un altra sezione del programma. Dato che QBasic dispone ora di sottoprogrammi e funzioni autonomi come alternative più strutturate rispetto alle subroutine, i programmatori si avvalgono raramente delle istruzioni GOSUB. In ogni caso questa parola chiave è ancora presente in QBasic, con lo scopo di mantenere la compatibilità con i programmi scritti con le versioni precedenti.

Sintassi

```
GOSUB {riga|etichetta}
.
.
.
RETURN [{riga|etichetta}]
```

riga o etichetta indicano la destinazione di un'istruzione GOSUB. Quindi, l'esecuzione procede fino a quando viene individuata l'istruzione RETURN; a questo punto il controllo ritorna all'istruzione che segue GOSUB, oppure al numero di riga o all'etichetta specificati nell'istruzione RETURN.

Utilizzo

```
GOSUB Aggiorna
```

Salta alla riga che presenta l'etichetta Aggiorna.

```
RETURN
```

Esce da una subroutine e restituisce il controllo all'istruzione successiva a GOSUB.

```
RETURN Menu
```

Esce da una subroutine e passa all'istruzione seguente l'etichetta Menu.

Descrizione

L'istruzione GOSUB indica a QBasic di eseguire un salto incondizionato al numero di riga o all'etichetta specificati. Il controllo viene trasferito alla riga di destinazione, indipendentemente dal risultato delle istruzioni eseguite in precedenza.

Questa istruzione non può essere utilizzata per passare all'interno o all'esterno di un sottoprogramma o di una funzione definita dall'utente, mentre invece GOSUB può essere sfruttata per passare all'interno di un sottoprogramma o di una funzione, a patto che l'esecuzione ritorni a un'istruzione contenuta nella procedura chiamante.

Tutte le variabili utilizzate dalla subroutine sono condivise dalle altre istruzioni presenti nella parte del programma che la comprende. Qualsiasi modifica apportata a queste variabili viene mantenuta quando la subroutine termina. Le subroutine possono contenere istruzioni GOSUB che richiamano altre subroutine (in questo caso si parla di *nidificazione*) o persino esse stesse (*ricorsione*), anche se il programma deve allocare uno spazio per lo stack sufficiente per il livello di nidificazione utilizzato (l'istruzione CLEAR, illustrata nel Capitolo 18, consente di specificare le dimensioni dello stack).

Se viene utilizzato RETURN con un numero di riga o un'etichetta per uscire da una subroutine nidificata, l'indirizzo di ritorno deve trovarsi nella stessa subroutine che l'ha richiamato. Se le subroutine sono nidificate su più livelli, è possibile ritornare soltanto al livello che ha richiamato la subroutine.

Avvertenze

Se RETURN viene utilizzato con un numero di riga o un'etichetta, è necessario prestare attenzione al fatto che le istruzioni successive alla riga di destinazione consentano il ritorno dalle subroutine richiamate da qualsiasi altra riga del programma.

Esempio

Una subroutine può avere più di un punto di uscita e l'istruzione RETURN può passare il controllo a qualsiasi punto all'interno del programma. Il programma d'esempio di seguito riportato comprende una subroutine che passa in modo selettivo alle diverse destinazioni, in base alle risposte dell'utente. Vengono tracciati dei disegni sullo schermo, fino a quando l'utente non preme un tasto.

```
    CLEAR : SCREEN 1: COLOR 0, 1: RANDOMIZE TIMER
100
    VIEW
    LINE (0, 0)-(319, 199), 1, BF 'disegna un riquadro colorato nelle
                                  'dimensioni dello schermo
    VIEW (32, 4)-(287, 186), 0, 2 'riduce le dimensioni dello schermo grafico
    CLS
    xLimite% = 253: xLimite% = 180
    segnoA% = 0: versione% = 1: x1% = 0: y1% = 0
    tasso% = INT(RND * 3) + 2       'imposta il tasso di cambiamento
    colori% = INT(RND * 3) + 1
110
    IF x1% <= 0 THEN x1% = 0: cambiamentoX% = segnoA%    'imposta la direzione
                                                         'del cambiamento
    IF y1% <= 0 THEN y1% = 0: cambiamentoY% = segnoA%
    IF x1% >= xLimite% THEN x1% = xLimite%: cambiamentoX% = -segnoA%
    IF y1% >= yLimite% THEN y1% = yLimite%: cambiamentoY% = -segnoA%
    IF segnoA% <= 1 THEN segnoA% = 1: segnoB% = 1
    IF segnoA% >= tasso% THEN segnoA% = tasso%: segnoB% = -1

    x2% = 255 - x1%: y2% = 181 - y1%
    IF versione% = 1 THEN
        LINE (x1%, y1%)-(x2%, y2%), colori%, B    'disegna il contorno di un
                                                  'riquadro
        LINE (x1%, y1%)-(x2%, y2%), 0      'disegna una linea diagonale nera
        LINE (x1%, y2%)-(x2%, y1%), 0
        versione% = 2
    ELSE
        LINE (x1%, y1%)-(x2%, y2%), 3, B  'disegna il contorno di un riquadro
        LINE (x1%, y1%)-(x2%, y2%), colori%  'disegna una linea diagonale
        LINE (x2%, y1%)-(x1%, y2%), colori%
        versione% = 1                                    'cambia i colori
        colori% = INT(RND * 3) + 1
    END IF
    x1% = x1% + cambiamentoX%
    y1% = y1% + cambiamentoY%
    segnoA% = segnoA + segnoB%
    GOSUB 900: GOTO 110
120
    SCREEN 0, 0, 0: WIDTH 80: CLS
END

900
    IF INKEY$ <> "" THEN LOCATE 25, 1 ELSE RETURN
    PRINT "U=Uscire  R=Ricominciare  C=Continuare";
    tasto$ = UCASE$(INPUT$(1))
    SELECT CASE tasto$
        CASE "U"
            RETURN 120
        CASE "R"
            RETURN 100
        CASE ELSE
            VIEW: LINE (0, 188)-(319, 199), 1, BF
            VIEW (32, 4)-(287, 186), , 2
        END SELECT
RETURN
```

SHARED

Si consulti anche

COMMON, DIM, FUNCTION, SUB

Scopo

A meno che non vengano passate come argomenti, le variabili utilizzate dai sottoprogrammi e dalle funzioni che l'utente definisce mediante le istruzioni SUB e FUNCTION sono locali a quelle procedure e le istruzioni nel codice a livello del modulo non possono accedervi. L'istruzione SHARED consente invece di rendere le variabili specificate disponibili sia alla procedura che al programma principale. Questo permette di restituire il risultato delle operazioni all'interno della procedura e consente alle funzioni di restituire più di un valore.

Sintassi

```
SHARED var [AS tipo] [, var [AS tipo]...]
```

var rappresenta una qualsiasi variabile di QBasic. Una variabile (all'infuori del tipo di default a precisione semplice) può comprendere un carattere di dichiarazione del tipo, può essere seguita dalla clausola AS con INTEGER, LONG, SINGLE, DOUBLE, STRING (compresa una stringa a lunghezza fissa), oppure può essere il nome di un tipo definito dall'utente. Se la variabile è un array deve essere seguita dalle parentesi, sebbene non sia necessario elencare il numero delle dimensioni.

Le istruzioni SHARED devono essere inserite all'inizio della procedura, prima di qualsiasi istruzione eseguibile.

Utilizzo

```
SHARED studente%, gradi!()
```

Consente che il contenuto di una variabile di tipo intero o di un array a precisione semplice sia disponibile per la procedura contenente l'istruzione SHARED.

Descrizione

Le istruzioni SHARED possono essere utilizzate soltanto nei sottoprogrammi e nelle funzioni definite dall'utente e indicano i nomi delle variabili presenti nel programma principale a cui deve accedere il sottoprogramma o la funzione. Queste variabili sono disponibili soltanto per quei sottoprogrammi e funzioni che comprendono istruzioni SHARED. Una variabile di una procedura che non è condivisa può avere lo stesso nome di una variabile contenuta nel programma principale; dato che non è visibile a quest'ultimo, QBasic considera la variabile della procedura diversa da quella del programma principale.

Il fatto di specificare le variabili in una istruzione SHARED elimina la necessità di passarle come parametri. Come con i parametri, qualsiasi modifica apportata a una variabile nella procedura che la condivide rimane valida quando il controllo ritorna al programma principale.

La visibilità delle istruzioni SHARED si estende soltanto alla procedura e al programma principale che le contengono.

Suggerimenti

Le variabili specificate possono essere rese disponibili a tutte le funzioni e i sottoprogrammi nel modulo, dimensionandole in un'istruzione DIM mediante la parola chiave SHARED nel codice a livello del modulo, come nell'esempio di seguito riportato.

```
DIM SHARED volume!, raggio!, diametro!
```

Ciò elimina la necessità di utilizzare istruzioni SHARED separate in tutte le procedure che devono accedere a quelle variabili.

Esempio

Oltre ai parametri e ai valori restituiti, talvolta si desidera che determinate procedure condividano determinate variabili standard definite nel codice a livello del modulo. Il programma di seguito riportato utilizza una funzione che permette all'utente di selezionare il nome di un file; la funzione utilizza l'istruzione SHARED per accedere alle variabili definite nel programma principale.

```
DECLARE FUNCTION OttieneFile$ (nomePercorso$)

CONST FALSO = 0, VERO = NOT FALSO               'le costanti sono globali
DIM SHARED termine AS INTEGER
cr$ = CHR$(13): esc$ = CHR$(27): backSpace$ = CHR$(8): curSinistra$ = CHR$(29)

nomeFile$ = OttieneFile$("*.*")
LOCATE 15, 1

IF termine THEN PRINT "Terminato!" ELSE PRINT " Hai scelto "; nomeFile$
END

FUNCTION OttieneFile$ (percorso$) STATIC
    SHARED cr$, esc$, backSpace$, curSinistra$   'condivisa con il programma
                                                 'principale
    termine = FALSO                         'condivisa con il programma principale
    buffer$ = "": puntatore% = 0
    CLS : FILES percorso$                   'stampa i nomi dei file

    PRINT "Inserisci il nome di un file: ";
    DO
        tasto$ = UCASE$(INKEY$)
        SELECT CASE tasto$
            CASE esc$
                 termine = VERO
                 EXIT DO
            CASE backSpace$
                 PRINT " "; curSinistra; curSinistra$;
                 puntatore% = puntatore% - 1
                 buffer$ = LEFT$(buffer$, puntatore%)
            CASE ""
            CASE ELSE
                 PRINT tasto$;
                 buffer$ = buffer$ + tasto$
                 puntatore% = puntatore% + 1
            END SELECT
        LOOP UNTIL tasto$ = CHR$(13)   'cicla finchè l'utente non preme Invio
        OttieneFile$ = buffer$
END FUNCTION
```

STATIC

Si consulti anche

COMMON, DIM, SHARED

Scopo

Per default, le variabili presenti in un sottoprogramma o in una funzione sono dinamiche, ovvero vengono ricreate ogniqualvolta la procedura viene richiamata. L'istruzione STATIC mantiene inalterato il contenuto delle variabili selezionate nella procedura, permettendo in questo modo di mantenerne i valori tra le chiamate. Questo consente di utilizzare i risultati dei calcoli nelle chiamate successive alla procedura.

Sintassi

```
STATIC var [AS tipo] [, var [AS tipo]...]
```

var rappresenta una qualsiasi variabile di QBasic. Una variabile (all'infuori del tipo di default a precisione semplice) può comprendere un carattere di dichiarazione del tipo, può essere seguita dalla clausola AS con INTEGER, LONG, SINGLE, DOUBLE, STRING (compresa una stringa a lunghezza fissa), oppure può essere il nome di un tipo definito dall'utente. Se la variabile è un array deve essere seguita dalle parentesi, sebbene non sia necessario elencare il numero delle dimensioni.

Utilizzo

```
STATIC contatore%, totale
```

Questa istruzione rende una variabile di tipo intero e una a precisione singola locali alla procedura e ne mantiene i valori anche dopo che la procedura è terminata.

Descrizione

A meno che non venga specificato diversamente, tutte le variabili denominate in una funzione o in un sottoprogramma sono dinamiche; QBasic le crea quando la procedura viene richiamata (generalmente per mantenere valori temporanei necessari ai compiti che la procedura deve eseguire), le distrugge e ne elimina il contenuto quando la procedura termina l'esecuzione. Questo tipo di variabile viene spesso denominata *automatica* è presente soltanto mentre è attiva la funzione o il sottoprogramma, ed è distrutta quando l'esecuzione termina, liberando in questo modo la memoria assegnata.

Talvolta si desidera mantenere il valore di una o più variabili tra le chiamate, in modo da poterle nuovamente utilizzare quando la procedura viene eseguita la volta successiva. Per eseguire questa operazione è sufficiente inserire all'inizio della procedura, prima di qualsiasi istruzione eseguibile, l'istruzione STATIC, seguita da un elenco di variabili i cui valori si desidera conservare.

Oppure, è possibile preservare tutte le variabili utilizzando la parola chiave STATIC nella prima riga della definizione della procedura, dopo il nome di quest'ultima e l'elenco degli argomenti. In questo modo, ogni variabile locale nella procedura diventa statica.

Le variabili statiche mantengono il proprio contenuto persino quando la procedura non è attiva. Dato che QBasic non deve crearle ogni volta che la procedura viene richiamata, l'utilizzo delle variabili statiche consente di velocizzare l'esecuzione del programma, anche se la memoria disponibile per altri tipi di elaborazione diminuisce.

Le variabili dinamiche sono locali al sottoprogramma o alla funzione che le contiene, dato che non esistono quando la procedura non è in esecuzione. Le variabili statiche sono anche locali: se una variabile avente lo stesso nome viene dimensionata in un'istruzione DIM con SHARED, oppure dichiarata in un'istruzione COMMON nel programma principale, QBasic la considera una variabile diversa. Nella stessa procedura non è possibile definire una variabile sia in un'istruzione SHARED che in una STATIC.

Sebbene l'istruzione STATIC venga spesso utilizzata nei sottoprogrammi e nelle funzioni, è anche possibile sfruttarla all'interno delle funzioni DEF FN multiriga nel codice a livello del modulo. In questo modo le variabili vengono dichiarate locali alla funzione DEF FN, e possono avere lo stesso nome delle variabili presenti nella parte restante del programma senza per questo interferire.

Avvertenze

L'istruzione STATIC non deve essere confusa con il metacomando $STATIC, che è un'istruzione per il compilatore che consente di allocare memoria per gli array, ma che non influenza l'esistenza o la visibilità delle variabili, una volta che il programma è iniziato.

Esempio

Il programma di seguito riportato dimostra che la variabile statica i% conserva il proprio valore tra le chiamate alla procedura Contatore che la contiene. Si noti che essa non viene influenzata dalla variabile dello stesso tipo presente nel programma principale. D'altro canto la variabile j% è dinamica, ovvero viene creata con il valore iniziale di 0 ogniqualvolta la procedura viene eseguita.

```
DECLARE SUB Contatore (cont AS INTEGER)

CLS
FOR i% = 0 TO 9
    j% = i%: PRINT
    Contatore i%
NEXT i%
END

SUB Contatore (cont AS INTEGER)
    STATIC i%
    PRINT USING "   # "; cont;
    FOR a% = 0 TO 9
        i% = i% + 1
        PRINT USING " ### "; i%;
    NEXT a%
    PRINT TAB(6);
    FOR a% = 0 TO 9
        j% = j% + 1
        PRINT USING " ### "; j%;
    NEXT a%
END SUB
```

SUB

Si consulti anche

CALL, DECLARE, DEF FN, EXIT, FUNCTION, SHARED, STATIC

Scopo

L'istruzione SUB definisce sottoprogrammi che possono essere richiamati da un punto qualsiasi all'interno del programma creato.

Sintassi

```
FUNCTION Nome [(arg[AS tipo][, (arg[AS tipo]...])] [STATIC]
   [SHARED var [, var...]]
   [STATIC var [, var...]]
   .
   .
   .
[EXIT SUB]
   .
   .
   .
END FUNCTION
```

Nome rappresenta un qualsiasi nome corretto di QBasic, che viene utilizzato per richiamare il sottoprogramma.

arg è il nome di una variabile passata al sottoprogramma come argomento (utilizzando un carattere di dichiarazione del tipo oppure la clausola AS con INTEGER, LONG, SINGLE, DOUBLE, STRING) o un tipo definito dall'utente. Gli array passati come argomenti devono comprendere le parentesi dopo il loro nome, sebbene non sia necessario specificare il numero delle dimensioni.

L'istruzione SHARED specifica le variabili presenti nel programma principale che devono essere utilizzate da sottoprogramma, ma che non vengono specificamente passate come argomenti.

L'istruzione STATIC specifica le variabili locali i cui valori vengono mantenuti tra le chiamate al sottoprogramma.

L'istruzione EXIT SUB restituisce il controllo al programma chiamante, prima che raggiunga il normale punto di uscita (l'istruzione END SUB).

Utilizzo

```
SUB CvtSegno (senzasegno&, consegno%)
   IF senzasegno& > 32767 THEN
      consegno% = senzasegno& - 65536
   ELSE
      consegno% = senzasegno&
   END IF
END SUB
```

Individua il valore equivalente con segno di 2 byte di senzasegno& (un intero lungo senza segno) e ne assegna il valore al parametro consegno%.

Descrizione

I sottoprogrammi sono procedure autonome che possono essere aggiunte ai programmi QBasic per eseguire un compito specifico. In modo analogo alle subroutine tradizionali, i sottoprogrammi possono essere richiamati da un punto qualsiasi del programma.

Se viene creato un sottoprogramma utilizzando l'editor di QBasic, viene immediatamente creata l'istruzione DECLARE corretta.

Per eseguire un sottoprogramma è necessario richiamarlo utilizzando il nome come oggetto della chiamata e inserendo i propri argomenti al posto di quelli riportati nell'elenco dei parametri, come nel seguente esempio:

```
CALL QuickSort(punteggio%(), LBOUND(punteggio%), UBOUND(punteggio%))
```

Si noti che gli argomenti non devono essere effettivamente variabili, dato che è anche possibile utilizzare nell'elenco dei parametri costanti ed espressioni, a patto che corrispondano al tipo di argomenti che il sottoprogramma prevede.

Se un'istruzione DECLARE per un sottoprogramma esiste nel modulo in uso, è possibile tralasciare la parola chiave CALL in una chiamata al sottoprogramma stesso. In questo caso, è necessario non inserire l'elenco dei parametri tra parentesi, come nell'esempio di seguito riportato.

```
QuickSort punteggio%(), LBOUND(punteggio%), UBOUND(punteggio%)
```

Tranne gli argomenti passati e le variabili specificate in un'istruzione SHARED, tutte le variabili sono locali al sottoprogramma. Questo significa che è possibile sfruttare qualsiasi nome di variabile senza interferire con le variabili che hanno lo stesso nome nel programma principale.

Al contrario delle funzioni, i sottoprogrammi non restituiscono un valore; tuttavia, dato che gli argomenti utilizzati vengono passati per riferimento, qualsiasi modifica apportata agli argomenti rimane valida anche quando il sottoprogramma termina.

Ricorsione

Una caratteristica importante dei sottoprogrammi (e delle funzioni) consiste nel fatto che possono essere ricorsivi, ovvero possono richiamare se stessi. Le procedure ricorsive utilizzano molto spazio nello stack, dato che, ogni volta che un programma viene richiamato, QBasic assegna una nuova serie di variabili locali e le memorizza nello stack del programma. Dato che la ricorsione richiede l'utilizzo di nuove variabile locali, la parola chiave STATIC non viene utilizzata nelle procedure ricorsive.

La ricorsione risulta molto spesso il metodo più efficace per la programmazione di applicazioni in cui devono essere elaborati blocchi di dati di ampie dimensioni. Il programma di seguito riportato utilizza la ricorsione per eseguire l'ordinamento di un array.

```
SUB QuickSort (array%(), inizio%, fine%)
    superiore% = inizio%: inferiore% = fine%
    comparazione% = array%((inizio% + fine%) \ 2)
    DO
        DO WHILE array%(superiore%) < comparazione% AND superiore% < fine%
            superiore% = superiore% + 1
```

```
          LOOP
          DO WHILE comparazione% < array%(inferiore%) AND inferiore% > inizio%
              inferiore% = inferiore% - 1
          LOOP
          IF inferiore% > superiore% THEN
              SWAP array%(superiore%), array%(inferiore%)
              superiore% = superiore% + 1: inferiore% = inferiore% - 1
          END IF
          IF inferiore% = superiore% THEN superiore% = superiore% + 1
      LOOP UNTIL superiore% > inferiore%
      IF inizio% < inferiore% THEN QuickSort array%(), inizio%, inferiore%
      'ricursione
      IF superiore% < fine% THEN QuickSort array%(), superiore%, fine%
      'ricursione
END SUB
```

Questo sottoprogramma utilizza un algoritmo di ordinamento di tipo detto *quicksort* per ordinare un array in base a un ordine ascendente. È necessario fornire il nome dell'array e il limite inferiore e superiore degli elementi da ordinare. Il sottoprogramma suddivide gli elementi in parti sempre più piccole e richiama se stesso in modo ricorsivo per ordinare ogni partizione fino a quando non viene ordinato tutto l'array.

Avvertenze

QBasic talvolta confonde le etichette di programma con le chiamate ai sottoprogrammi che non assumono parametri. Questo si verifica soltanto se la chiamata è la prima istruzione sulla riga e non comprende la parola chiave CALL, come nel seguente esempio:

```
StampaSchermata:
```

In questo caso QBasic potrebbe considerare l'istruzione come etichetta di riga oppure come chiamata a un sottoprogramma; non viene visualizzato alcun messaggio di errore, tuttavia, dato che il programma non viene mai eseguito, per individuarne la ragione effettiva potrebbe essere necessario molto tempo.

Per impedire tutto ciò, è sufficiente utilizzare in questi casi un'istruzione CALL esplicita, anche se il modulo comprende un'istruzione DECLARE per il sottoprogramma.

Esempio

I programmatori creano spesso sottoprogrammi che possono essere sfruttati in diversi programmi. Il programma QBasic di seguito riportato utilizza un sottoprogramma per fornire un gestore standard di menu; il programma chiamante fornisce l'elenco delle opzioni (come array di stringhe) e la posizione dello schermo in cui deve essere tracciato il menu. Il sottoprogramma traccia il menu, gestisce l'imput dell'utente e restituisce l'opzione selezionata.

```
DECLARE SUB VerMenu (riga%, colonna%, menu%)

CONST FALSO = 0, VERO = NOT FALSO

DIM SHARED menu$(0 TO 9): menu% = 8
CLS : RESTORE

FOR i% = 1 TO menu%                          'ottiene le opzioni del menu
```

```
    menu$(0) = menu$(0) + CHR$(i% + 64)
    READ menu$(i%)
    menu$(i%) = CHR$(i% + 64) + ". " + menu$(i%)
    menu$(i%) = LEFT$(menu$(i%) + SPACE$(24), 24)
NEXT i%

VerMenu 8, 28, menu%

LOCATE 20, 10, 1
IF termine% THEN
    PRINT "TERMINATO!"
ELSE
    PRINT "Opzione"; scelta%; "selezionata"
END IF
END

DATA "Impostazioni attuali","Leggi posta personale"
DATA "Leggi messaggi","Spedisci un messaggio","Download di un file"
DATA "Upload di un file","Chiama il Sysop","Arrivederci - log off"



SUB VerMenu (riga%, colonna%, menu%) STATIC
    SHARED termina%, scelta%

    termina% = FALSO: scelta% = 1
    FOR i% = 1 TO menu%
        LOCATE riga% + i%, colonna%, 0
        COLOR 15, 0: PRINT LEFT$(menu$(i%), 1);
        COLOR 7, 0: PRINT MID$(menu$(i%), 2);
    NEXT i%
    DO
        COLOR 14, 4
        lineaAltaInt% = riga% + scelta%
        LOCATE lineaAltaInt%, colonna%: PRINT menu$(scelta%);
        DO
            tasto$ = INKEY$
        LOOP WHILE tasto$ = ""
        ultimaScelta% = scelta%
        IF LEN(tasto$) < 2 THEN
            IF tasto$ = CHR$(13) THEN EXIT DO
            IF tasto$ = CHR$(27) THEN termine% = VERO: EXIT DO
            tasto$ = UCASE$(tasto$)
            m% = INSTR(menu$(0), tasto$)
            IF m% > 0 THEN scelta% = m%
        ELSE
            scanCodice% = ASC(MID$(tasto$, 2, 1))
            IF scanCodice% = 80 THEN scelta% = scelta% + 1
            IF scanCodice% = 72 THEN scelta% = scelta% - 1
        END IF
        IF scelta% > menu% THEN scelta% = 1
        IF scelta% < 1 THEN scelta% = menu%
        IF scelta% <> ultimaScelta% THEN
             LOCATE riga% + ultimaScelta%, colonna%
             COLOR 15, 0: PRINT LEFT$(menu$(ultimaScelta%), 1);
             COLOR 7, 0: PRINT MID$(menu$(ultimaScelta%), 2);
        END IF
    LOOP WHILE 1
    COLOR 7, 0
END SUB
```

Capitolo 5

Stringhe

INTRODUZIONE

Sebbene i computer vengano spesso considerati "mangiatori di numeri", la maggior parte dei dati che elaborano sono caratteri di lettere e di prospetti, etichette di fogli elettronici, nomi e indirizzi di elenchi di corrispondenza e così via.

Il modo in cui QBasic interpreta i dati in un byte o insieme di byte in memoria dipende dal tipo di dati che viene rappresentato. Se il programma deve gestire dei numeri vengono utilizzate le variabili numeriche, ad esempio totale% e imposta! in cui i dati contenuti in ogni byte della variabile sono parte di un numero binario. Il tipo di variabile definito indica in che modo QBasic raggruppa i byte in memoria e come ne interpreta i relativi valori. Ad esempio, totale% è un intero di 2 byte, mentre imposta! è un intero di 4 byte, a virgola mobile, a precisione semplice; se ad esempio il valore di un byte particolare in memoria è 65, il valore della variabile che contiene quel byte dipende dal suo tipo. Se la variabile è un intero di 2 byte, il suo valore dipende da quello dei 2 byte.

D'altro canto, se viene definita una variabile stringa (ad esempio Nome$), ogni byte viene considerato come carattere ASCII con un valore compreso nell'intervallo da 0 a 127, oppure come un carattere esteso IBM compreso nell'intervallo da 128 a 255. Quindi, se una variabile stringa comprende il valore 65 in un byte, esso viene interpretato come una A maiuscola, anziché come parte di un numero, come riportato nella Figura 5.1. Il set di caratteri ASCII e il set di caratteri estesi IBM contengono 256 caratteri diversi, ognuno dei quali corrisponde a un determinato valore numerico. Il set di caratteri ASCII comprende le lettere maiuscole e minuscole, le dieci cifre, i segni di punteggiatura, i simboli matematici e i caratteri di controllo della stampante (ad esempio il ritorno a capo, il codice di fine riga e quello di cambio pagina). Il set di caratteri estesi IBM comprende i caratteri grafici, i simboli matematici e i caratteri relativi alle lingue straniere non presenti nella lingua inglese.

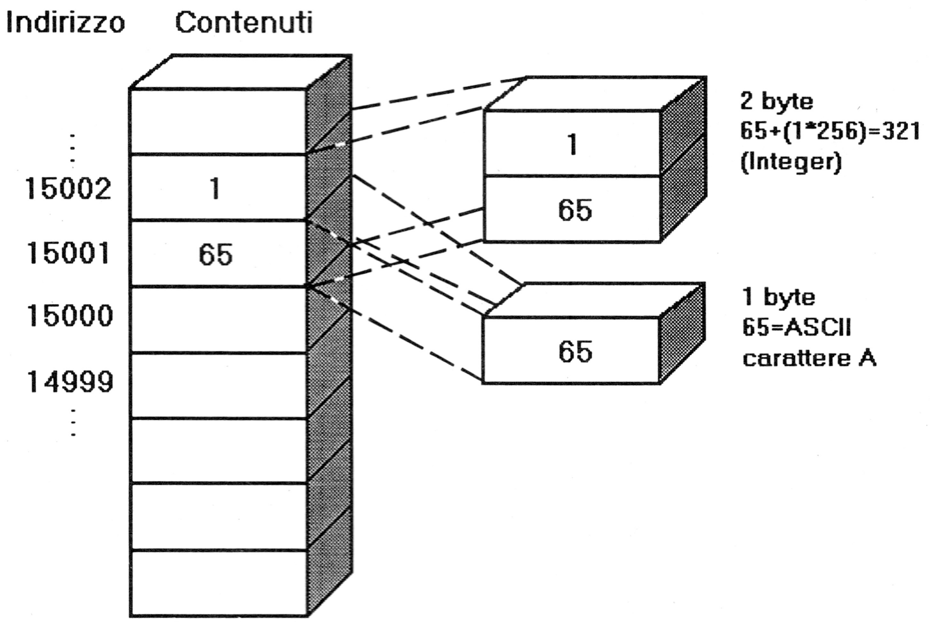

Figura 5.1 La rappresentazione di un valore come numero o come carattere

Una *stringa* è semplicemente una serie di caratteri, come "QBasic" o CHR$(7) + CHR$(12). Nel primo esempio i doppi apici racchiudono una stringa letterale, ovvero i caratteri che costituiscono la parola "QBasic", mentre nel secondo esempio la funzione CHR$ restituisce il carattere corrispondente al valore ASCII indicato; in questo caso la stringa è composta dal carattere ASCII 7 (che causa l'emissione di un suono acustico) e dal carattere ASCII 12 (che provoca un salto pagina oppure la cancellazione dello schermo).

QBasic presenta molte istruzioni e funzioni, riportate in ordine alfabetico nella Tabella 5.1, che consentono di gestire le stringhe.

DEFINIZIONE E UTILIZZO DELLE STRINGHE

Le stringhe si prestano a molti usi: è possibile definire variabili o costanti che contengono valori stringa; memorizzare le stringhe in un array; passarle o riceverle come parametri dalle funzioni e dai sottoprogrammi e definire tipi di dati stringa in base alle proprie necessità.

Tabella 5.1 Le istruzioni e le funzioni per la gestione delle stringhe di QBasic

Istruzione o funzione	Scopo
ASC	Restituisce il codice ASCII (oppure il codice esteso IBM) d un carattere o del primo carattere di una stringa
CHR$	Restituisce il carattere corrispondente al codice ASCII indicato (oppure al codice esteso IBM)
INSTR	Restituisce la posizione all'interno di una stringa, in corrispondenza della quale inizia una determinata stringa
LCASE$	Restituisce una stringa in caratteri minuscoli
LEFT$	Restituisce il numero specificato di caratteri, partendo dalla parte sinistra di una stringa
LEN	Restituisce la lunghezza di una stringa
MID$ (Funzione)	Restituisce il numero specificato di caratteri, partendo da una determinata posizione della stringa
MID$ (Istruzione)	Sostituisce i caratteri in una stringa, partendo da una determinata posizione, con i caratteri specificati
RIGHT$	Restituisce il numero specificato di caratteri, partendo dalla parte destra di una stringa
SPACE$	Restituisce una stringa costituita dal numero specificato di spazi
STR$	Restituisce una stringa contenente la rappresentazione del valore specificato come stringa
STRING$	Restituisce una stringa contenente un carattere ripetuto il numero specificato di volte
UCASE$	Restituisce una stringa in lettere maiuscole
VAL	Restituisce il valore numerico rappresentato dalle cifre e dagli altri caratteri speciali della stringa specificata

Variabili stringa

Per definire una variabile stringa è necessario scegliere un nome che ne descriva il contenuto. Ad esempio, una variabile che contiene il nome di una città da stampare sull'etichetta di una lettera circolare potrebbe essere chiamata citta$; il simbolo di dollaro ($) indica che quella variabile contiene dati stringa. Per assegnare un valore alla stringa è sufficiente utilizzare il segno di uguale (=) seguito da un'espressione stringa, che può contenere una stringa (tra doppi apici), una chiamata a una funzione di stringa, il nome di un'altra variabile stringa oppure una combinazione di questi elementi.

É anche possibile utilizzare l'istruzione DEFSTR per indicare che tutte le variabili il cui nome inizia con una lettera compresa in un intervallo particolare sono per default variabili stringa, come nel programma di seguito riportato.

```
DEFSTR A-Z
citta = "Milano"
PRINT citta$
```

In questo esempio, QBasic considera citta e citta$ come lo stesso identificatore, tuttavia citta! è ancora una variabile numerica a precisione semplice.

È anche possibile definire il tipo di una variabile come stringa utilizzando l'istruzione DIM seguita da AS STRING, ad esempio:

```
DIM Nome AS STRING
```

definisce Nome come una variabile stringa.

L'utilizzo di DIM il metodo più complicato per denominare una variabile, ma consente di dichiararne il tipo prima di utilizzarla.

Costanti stringa

La parola chiave CONST consente di definire una costante stringa. La differenza tra quest'ultima e una stringa normale consiste nel fatto che dopo aver definito una costante stringa non è possibile modificarne il contenuto; ad esempio l'istruzione:

```
CONST LINEETTE$= "---"
```

definisce una stringa costituita da tre trattini. Se viene successivamente inserita un'istruzione che cerca di modificare il valore di LINEETTE$, QBasic visualizza il messaggio di errore: "Duplice definizione".

È anche possibile creare costanti numeriche che contengono il valore di determinati caratteri, come nel seguente esempio.

```
CONST CP = 12 'valore ASCII per cambio pagina
CONST FAIBEEP = 7 'valore ASCII per un beep
```

A questo punto è possibile immetere la stringa CHR$(7) + CHR$(12) come CHR$(FAIBEEP) + CHR$(CP) (la parola chiave BEEP non può essere utilizzata, poiché è una parola riservata di QBasic). Dato che una costante non può contenere una chiamata a una funzione, non è possibile definire una constante stringa FAIBEEP nel seguente modo:

```
CONST FAIBEEP$ = CHR$(7)
```

Array di tipo stringa

Per creare un array di tipo stringa è sufficiente utilizzare il nome di una variabile stringa nell'istruzione DIM; ad esempio:

```
DIM giorni$(7)
```

specifica un array denominato giorni$(7) che contiene sette stringhe. L'esempio di seguito riportato utilizza questo array.

```
DIM giorni$(7)
giorni$(1) = "Domenica": giorni$(2) = "Lunedi"
giorni$(3) = "Martedi": giorni$(4) = "Mercoledi"
giorni$(5) = "Giovedi": giorni$(6) = "Venerdi"
giorni$(7) = "Sabato"
```

```
FOR giorno% = 1 TO 7
   PRINT giorni$(giorno%); "";
NEXT giorno%
```

Questo programma visualizza i giorni della settimana.

Stringhe a lunghezza variabile e memorizzazione delle stringhe

La maggior parte delle versioni di Basic supporta soltanto le stringhe che possono variare in lunghezza. In effetti, tutte le variabili stringa di QBasic utilizzate finora possono contenere un massimo di 32767 caratteri; esse vengono memorizzate (insieme agli array statici, alle normali variabili numeriche e allo stack) in un segmento di memoria di 32 KB denominato DGROUP, l'area dati di default. QBasic crea un descrittore stringa di 4 byte per ogni stringa a lunghezza variabile: i primi due contengono l'indirizzo della stringa all'interno di DGROUP, mentre i 2 byte successivi contengono la lunghezza della stringa. Nella Figura 5.2 è illustrata la modalità di memorizzazione delle stringhe a lunghezza variabile.

La separazione del descrittore dal contenuto permette a QBasic di spostare in modo più efficiente le stringhe in memoria, dato che l'indirizzo nel descrittore viene aggiornato soltanto quando è necessario.

Avendo ben chiara la modalità utilizzata da QBasic per memorizzare le stringhe, è possibile gestire la memoria disponibile nei programmi che utilizzano (e riutilizzano) molte stringhe e array di tipo stringa.

Se si modifica il valore di una stringa mediante l'istruzione di assegnazione oppure richiamando una o più delle funzioni di stringa descritte in questo capitolo, QBasic alloca la memoria necessaria modificando l'indirizzo nel descrittore e aggiornando la lunghezza. Infine, la memoria occupata dai valori stringa che non vengono più utilizzati viene recuperata mediante il processo di "recupero e raggruppamento delle posizioni inutilizzate" (garbage collection).

Dato che QBasic esegue continuamente questo processo, la memoria che contiene stringhe ormai obsolete viene liberata non appena il programma ne modifica il contenuto.

Stringhe a lunghezza fissa

QBasic dispone di un altro tipo di stringa che ha una lunghezza fissa, indipendentemente dal numero di caratteri a essa assegnati. Per dichiarare una stringa a lunghezza fissa, è sufficiente utilizzare l'istruzione di seguito riportata.

```
DIM stringaFissa AS STRING * 10
```

Non è possibile utilizzare il suffisso $; il qualificatore AS STRING definisce il tipo di variabile. È invece necessario che, dopo il qualificatore, venga inserito l'asterisco (*) seguito dalla lunghezza della stringa in byte (al massimo 32767). Il numero di byte in una stringa è uguale al numero di caratteri.

Lo svantaggio legato all'utilizzo delle stringhe a lunghezza fissa consiste nel fatto che una parte di spazio viene generalmente sprecato, dato che normalmente il valore

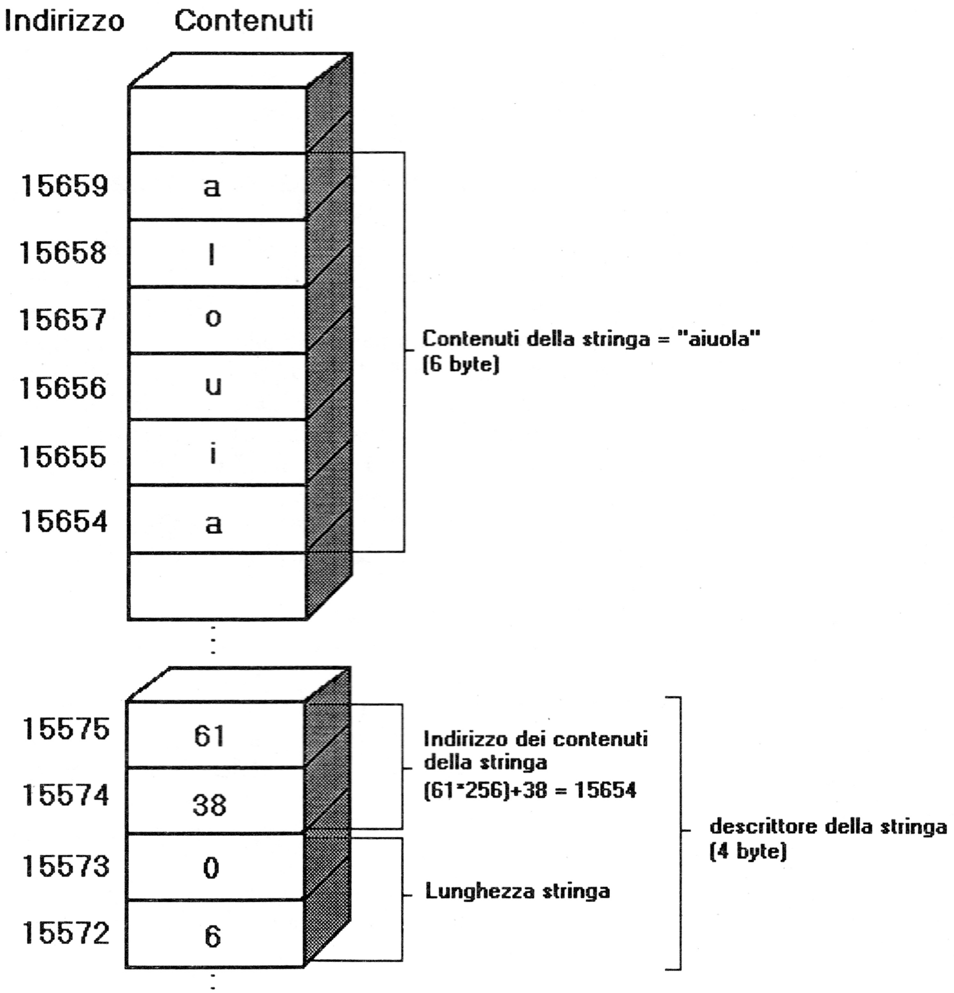

Figura 5.2 La memorizzazione delle stringhe a lunghezza variabile

effettivo di stringa memorizzato nella variabile non occupa totalmente lo spazio allocato. Inoltre, non è possibile memorizzare una stringa di lunghezza superiore a quella specificata.

Il valore assegnato alla stringa a lunghezza fissa è giustificato a sinistra, mentre tutte le posizioni non occupate a destra sono riempite con spazi. L'istruzione RSET può essere sfruttata per assegnare a una variabile un valore giustificato a destra nella stringa e con alcuni spazi iniziali. Per quanto riguarda le stringhe a lunghezza fissa, la funzione LEN restituisce la lunghezza definita, non quella effettiva.

FUNZIONI DI GESTIONE DELLE STRINGHE

Confronti

Le stringhe non possono essere verificate e confrontate come i valori numerici. Una stringa è "maggiore" di un'altra stringa se il primo carattere diverso ha un valore ASCII superiore (ad esempio la lettera A ha un valore ASCII inferiore rispetto alla lettera Z, quindi Z è maggiore di A). La Tabella 5.2 contiene alcuni esempi di confronti di stringa.

Questi esempi illustrano i seguenti punti.

- I confronti che hanno successo restituiscono il valore -1.
- I confronti che non hanno successo restituiscono il valore 0.
- Per verificare se una stringa ha un valore (non è nulla), è necessario controllare il risultato della funzione LEN. Una stringa nulla ha lunghezza 0, quindi il risultato di un test di una stringa nulla è falso.
- I confronti tengono conto delle lettere maiuscole e minuscole, queste ultime seguono (sono superiori) le lettere maiuscole.
- Se tutti i caratteri in una stringa più breve corrispondono ai relativi caratteri in una stringa più lunga, quest'ultima è maggiore di quella più breve.

Tabella 5.2 Alcuni esempi di confronto tra stringhe

Confronto	Risultato
a$ = "QBasic"	
b$ = "Programmazione"	
a$>b$	Vero(-1)
a$<b$	Falso(0)
DO WHILE LEN(A$)	Vero(-1)
LOOP	
a$ = ""	
LEN(a$)	Falso(0)
a$ = "microsoft"	
b$ = "Microsoft"	
a$>b$	Vero(-1)
a$ = "Q"	
b$ = "QBasic"	
b$>a$	Vero(-1)
b$<>a$	Vero(-1)

Estrazione di parti di una stringa

Per un confronto o una formattazione, molto spesso è necessario esaminare soltanto una parte di una stringa. La funzione LEFT$ può essere utilizzata per restituire un numero specificato di caratteri partendo dal lato sinistro di una stringa. Ad esempio, se la variabile linguaggio$ è uguale a "QBasic", la seguente istruzione:

```
PRINT LEFT$(linguaggio$, 1)
```

visualizza "Q". Si noti che tutte le funzioni che operano sulle stringhe restituiscono una copia modificata della stringa, ovvero la stringa originale rimane invariata.

La funzione RIGHT$ può invece essere utilizzata per esaminare i caratteri, partendo dal lato destro della stringa. Ad esempio l'istruzione:

```
RIGHT$(linguaggio$, 5)
```

restituisce la stringa "Basic".

Le funzioni RTRIM$ e LTRIM$ consentono di eliminare gli spazi iniziali e finali da una stringa a lunghezza fissa. L'eliminazione di questi spazi impedisce salti non voluti nelle stampe e inoltre risolve i problemi nel caso di confronti tra una stringa a lunghezza fissa e una a lunghezza variabile. Si consideri il seguente esempio:

```
a$ = "QBasic"              'stringa lunghezza variabile
DIM b AS STRING * 10       'stringa a lunghezza fissa
b = "QBasic"               'stringa giustificata a sinistra
PRINT (<b> a$)             'confronta le stringhe
```

Dato che b comprende alcuni spazi finali che ampliano la stringa a lunghezza fissa fino a dieci caratteri, b è "maggiore di" a$, anche se il relativo contenuto è identico. Tuttavia, se viene utilizzata l'istruzione:

```
PRINT (RTRIM$(b) > a$)
```

invece della quarta riga dell'esempio precedente, allora b non è "maggiore di" a$.

La funzione MID$ può essere utilizzata per estrarre uno o più caratteri da qualsiasi posizione in una stringa; è necessario indicare la stringa, la posizione in corrispondenza della quale deve iniziare l'estrazione e il numero di caratteri da estrarre. Ad esempio, se a$ è uguale a "Tre topolini ciechi", la chiamata:

```
MID$(a$, 5, 8)
```

restituisce "topolini".

La funzione MID$ risulta forse più utile quando si tratta di eseguire un ciclo in una stringa, esaminando ogni carattere uno alla volta, come riportato nelle seguenti istruzioni:

```
a$ = "Tre topolini ciechi"
FOR car% = 1 TO LEN(a$)
   questoCar$ = MID$(a$, car%, 1)
   IF questoCar$ = " " THEN
      PRINT 'passa alla riga successiva
   ELSE
      PRINT questoCar$; 'stampa il carattere
   END IF
NEXT car%
```

L'esempio precedente consente di estrarre ogni carattere con la funzione MID$, utilizzando la posizione indicata dal contatore di ciclo car%. Se il carattere è uno spazio, l'output viene spostato alla riga successiva; in caso contrario, viene stampato il carattere attuale nella posizione successiva sulla riga.

Individuazione dei caratteri in una stringa

Molto spesso è necessario controllare se un carattere o una stringa sono contenuti in un'altra stringa. La funzione INSTR consente di ricercare una stringa all'interno di un'altra: accetta due stringhe come parametri e restituisce la posizione della seconda all'interno della prima. Se la seconda stringa non è contenuta all'interno della prima, INSTR restituisce 0, fornendo un valore falso per qualsiasi test, ad esempio:

```
a$ = "QBasic"
b$ = "Basic"
c$ = "Q"
d$ = "Pascal"
PRINT INSTR(a$, b$)      'visualizza 2
PRINT INSTR(a$, c$)      'visualizza 1
IF INSTR(a$, d$) THEN PRINT "Pascal è in Basic" 'non visualizza nulla
```

Nella prima istruzione PRINT, la funzione INSTR restituisce il valore 2, dato che "Basic" inizia in corrispondenza del secondo carattere nella stringa "QBasic". Nella seconda istruzione PRINT, INSTR restituisce il valore 1, dato che la posizione 1 in "QBasic" è "Q". La clausola THEN, presente nell'istruzione IF nell'esempio precedente, non viene eseguita, dato che INSTR restituisce 0 (ovvero, la stringa "Pascal" non è contenuta nella stringa "QBasic"). Si noti che, se dovesse esistere una corrispondenza in un punto qualsiasi della stringa, il valore restituito sarebbe superiore a 0; dato che QBasic considera qualsiasi valore diverso da 0 come vero, è possibile utilizzare INSTR con qualsiasi istruzione che richieda una condizione (IF, WHILE e così via); la condizione è vera soltanto se viene individuata la stringa.

INSTR può anche contenere un primo parametro facoltativo che indica la posizione, nella prima stringa, in corrispondenza della quale INSTR inizia a ricercare la seconda stringa. Ad esempio, l'istruzione :

```
INSTR(4, "uno più uno fa due", "uno")
```

restituisce il valore 9, dato che la ricerca inizia dopo il primo "uno" e restituisce la posizione del secondo "uno".

Concatenazione di stringhe

QBasic consente di congiungere due stringhe (unendole con l'operatore +) e di assegnarle a una variabile stringa. Questa operazione viene definita *concatenazione.*

Riempimento di stringhe

Due funzioni di QBasic, SPACE$ e STRING$ permettono di creare stringhe "piene" che possono essere utilizzate per formattare copie su carta. La funzione

SPACE$ consente di generare il numero richiesto di spazi, ad esempio l'istruzione:

```
pieno$ = SPACE$(10)
```

assegna un valore di 10 spazi alla stringa pieno$.

STRING$ è una funzione di carattere più generale, che restituisce una stringa contenente un carattere ripetuto un determinato numero di volte, ad esempio:

```
PRINT STRING$(10, "*")    'visualizza **********
PRINT STRING$(10, " ")    'analogo a SPACE$(10)
```

Le lettere maiuscole e minuscole

La funzione UCASE$ restituisce una copia di una stringa con tutte le lettere minuscole convertite in maiuscole, mentre la funzione LCASE$ esegue l'operazione opposta:

```
PRINT LCASE$("QBasic")    'stampa qbasic
PRINT UCASE$("QBasic")    'stampa QBASIC
```

I caratteri non alfabetici, quali i numeri e i segni di punteggiatura, non vengono influenzati dalle funzioni UCASE$ e LCASE$.

Modifica di una parte di stringa

È già stato illustrato in che modo la funzione MID$ permetta di individuare la posizione di un particolare carattere o di più caratteri in una stringa. QBasic fornisce inoltre l'istruzione MID$, che consente di sostituire un carattere o un gruppo di caratteri con un altro. Ad esempio, le due coppie di istruzioni di seguito riportate visualizzano le stringhe geodetico, geocentrico, geotermico.

```
a$ = "geodetico"
PRINT a$

MID$(a$, 4) = "centrico"
PRINT a$

a$ = "geodetico" 'ripristina il valore originale
MID$(a$, 4) = "termico"
PRINT a$
```

I caratteri nella stringa assegnata sostituiscono le rispettive parti nella stringa che viene specificata nel primo parametro a MID$, partendo dalla posizione fornita nel secondo parametro. Quindi l'istruzione:

```
MID$(a$, 4) = "centrico"
```

sostituisce i caratteri in a$ con i caratteri della stringa "centrico", partendo dal quarto carattere (la "d" di detico). Se si desidera eliminare una parte della stringa in fase di sostituzione, è sufficiente inserire alcuni spazi nella stringa di sostituzione. Dato che MID$ non può sostituire i caratteri presenti dopo l'ultimo carattere nella stringa originale, è necessario inserire in quest'ultima un numero sufficiente di spazi in modo

da poter inserire la stringa di sostituzione più lunga. E anche possibile indicare che soltanto una parte della stringa di sostituzione deve essere utilizzata.

Si noti che, al contrario delle funzioni di gestione delle stringhe, che restituiscono copie modificate di una stringa, l'istruzione MID$ modifica la stringa originale.

Conversione dei numeri in stringhe

In questa parte viene esaminata la rappresentazione di un valore come numero o come carattere (stringa). Per QBasic, un numero e una stringa sono due oggetti completamente diversi, nondimeno, spesso è necessario convertire una stringa in un numero e viceversa. In molte applicazioni si desidera ottenere un numero da un utente sotto forma di stringa, in modo da poterne controllare la validità, convertendolo quindi in un numero per un'ulteriore elaborazione (ad esempio calcoli). Inoltre, quando i numeri vengono visualizzati sullo schermo o stampati in un prospetto, talvolta si desidera convertirli nuovamente in stringhe, in modo da poterli facilmente formattare.

La funzione VAL può essere utilizzata per restituire il valore numerico di una stringa. La funzione STR$ può invece essere sfruttata per restituire la stringa che rappresenta un valore numerico, ad esempio le istruzioni:

```
PRINT VAL("12.5")
PRINT VAL("9999999999999999")
PRINT STR$(12.5)
PRINT STR$(1.5D8)
```

visualizzano il seguente output:

```
12.5
1D+16
12.5
150000000
```

La funzione VAL interpreta la stringa come se fosse stata digitata da tastiera sotto forma di numero; il risultato è generalmente un numero con le stesse cifre della stringa. Tuttavia, in alcuni casi, come nel secondo esempio, QBasic applica le proprie regole per la rappresentazione di numeri di ampie dimensioni. Dato che il numero specificato era troppo grande per poter essere rappresentato in forma decimale, QBasic lo ha convertito in una notazione esponenziale (scientifica) e arrotondato fino a 10 alla sedicesima potenza. STR$ converte generalmente le cifre (e qualsiasi segno o punto decimale) in caratteri in una stringa; tuttavia nell'ultimo esempio, dato che il numero si trova all'interno dell'intervallo che può essere rappresentato in forma decimale, 1.5D8 viene convertito in 150000000.

CONCLUSIONI

Le funzioni stringa di QBasic rappresentano gli strumenti più validi e maggiormente utilizzati. QBasic contiene altre funzioni stringa, che vengono esaminate nel seguito, alcune eseguono le stesse operazioni delle funzioni stringa trattate in questo capitolo. In molti casi è possibile eseguire un lavoro in molti modi diversi, soltanto la pratica è in grado di indicare quello che funziona meglio.

BibliOgrafia

Kernighan, Brian, and P.J. Plauger. *Software Tools*. Reading, Mass.: Addison-Wesley, 1976.

Prata, Stephen, with Harry Henderson. *The Waite Group's Microsoft QuickBasic Primer Plus*. Redmond, Wash.: Microsoft Press, 1990

ASC

Si consulti anche

CHR$, STRING$, VAL

Scopo

La funzione ASC restituisce il valore ASCII o quello del codice esteso IBM (nell'intervallo da 0 a 255) relativo al primo carattere della stringa.

Sintassi

```
ASC(stringa)
```

stringa può essere una qualsiasi variabile stringa, costante stringa, letterale di tipo stringa o espressione stringa.

Utilizzo

```
PRINT ASC("a")
```

Stampa il valore ASCII della lettera "a", ovvero 97.

```
PRINT ASC("Alto")
```

Stampa il valore ASCII della lettera "A", ovvero 65.

Descrizione

La funzione ASC restituisce il valore ASCII o quello del codice esteso IBM relativo al carattere indicato in un letterale di tipo stringa, in una costante stringa o in una variabile stringa. Se viene utilizzata una stringa contenente più di un carattere, ASC restituisce il valore del primo.

La funzione CHR$ funziona in modo opposto ad ASC, dato che restituisce il carattere di cui è stato specificato il valore ASCII.

Errori

Se alla funzione ASC viene fornita una stringa nulla (lunghezza 0) come argomento, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita". Se il valore fornito non è una stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato consente di creare un stringa codificata, aggiungendo il valore ASCII di ogni carattere del messaggio al valore del carattere corrispondente nella stringa del codice di tasto. La maggior parte dei caratteri risultanti fanno parte del set di caratteri esteso IBM e non sembrano per nulla oscuri. Come esercitazione, è possibile scrivere un programma che decodifica la stringa codificata da questo programma; nella Figura 5.3 è riportato un esempio dell'output.

```
CLS
INPUT "Testo del messaggio "; testo$

DO
   INPUT "Chiave del codice "; chiave$
   IF LEN(chiave$) < LEN(testo$) THEN
      PRINT "La chiave deve essere lunga almeno quanto il testo del messaggio."
   ELSE EXIT DO
   END IF
LOOP

FOR carattere% = 1 TO LEN(testo$)
    valTesto = ASC(MID$(testo$, carattere%, 1))   'valore del carattere del
                                                  'messaggio
    valCodice = ASC(MID$(chiave$, carattere%, 1)) 'valore del carattere del
                                                  'messaggio
    MID$(testo$, carattere%, 1) = CHR$(valCodice + valTesto)  'addiziona i
                                                              'valori
NEXT carattere%

PRINT "Il testo codificato è: "; testo$
```

```
Testo del messaggio?Questo e' un messaggio segreto.
Chiave del codice?Questo e' il codice segreto del messaggio segreto.
Il testo codificato è: ó_-µ__@-N@_+@-++_-¦ç_+ç_-_ßà++Ü
```

Figura 5.3 L'output di un programma di esempio per ASC

CHR$

Si consulti anche

ASC, SPACE$, STRING$

Scopo

La funzione CHR$ restituisce il carattere ASCII o quello esteso IBM corrispondente al valore del codice ASCII specificato. Questa funzione può essere utilizzata per inviare caratteri speciali di controllo alla stampante oppure per disegnare oggetti utilizzando il set di caratteri estesi IBM.

Sintassi

```
CHR$(valore)
```

valore è il valore del carattere ASCII o del carattere esteso IBM che si desidera utilizzare e deve trovarsi nell'intervallo tra 0 e 255.

Utilizzo

```
LPRINT CHR$(12)
```

Invia alla stampante un carattere di cambio pagina.

```
PRINT CHR$(205)
```

Visualizza il simbolo =.

```
a$ = "apici"
PRINT CHR$(34)+a$+CHR$(34)
```

Visualizza "apici" (compresi i doppi apici).

Descrizione

Esistono molti caratteri che non è facile immettere da tastiera, inserendoli in una stringa, quali i caratteri di controllo e i caratteri estesi IBM (i caratteri con valore ASCII di 128 e oltre). La funzione CHR$ consente di creare tutti i 256 caratteri ASCII e quelli del set di caratteri estesi IBM. La funzione ASC svolge un'operazione opposta, dato che restituisce il valore ASCII o quello del codice esteso IBM corrispondente al primo carattere della stringa specificata.

Commenti

Per creare degli spazi, la funzione SPACE$ è più semplice e rapida rispetto all'utilizzo di un ciclo che stampi ripetutamente CHR$(32) oppure la stringa " ". Se si desidera creare una stringa costituita da un carattere particolare ripetuto per un dato numero di volte, è sufficiente utilizzare la funzione STRING$.

Errori

Se il valore specificato con la funzione CHR$ non rientra nell'intervallo da 0 a 255, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita". Se il valore è una variabile stringa o un letterale di tipo stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi" (se viene specificato un carattere non numerico o una stringa senza doppi apici, il compilatore non genera alcun messaggio di errore e la funzione CHR$ si limita a restituire un carattere nullo, oppure il valore ASCII 0).

Esempio

L'esempio di seguito riportato utilizza CHR$ per memorizzare tre diversi caratteri di riempimento dei riquadri in un array di tipo stringa. All'utente viene richiesto di inserire il nome e il numero dei tre articoli venduti, mentre il programma genera un istogramma che rispecchia tali valori.

```
DIM barra$(3)
barra$(1) = CHR$(176): barra$(2) = CHR$(177): barra$(3) = CHR$(178)'tre barre

maxArticolo = 3

TYPE TipoArticolo                               'tipo di dati per ogni articolo
    nomeArticolo AS STRING * 10
    vendita AS INTEGER
END TYPE

DIM vendite(3) AS TipoArticolo            'array per gestire i nomi e i numeri
                                          'delle vendite
CLS
FOR articolo = 1 TO maxArticolo
    PRINT "Nome dell'articolo #"; articolo;
    INPUT vendite(articolo).nomeArticolo
    INPUT "Quanti ne sono stati venduti "; vendita
    vendite(articolo).vendita = vendita / 1000
NEXT articolo

'intestazione per il grafico
PRINT : PRINT "          Numero di articoli venduti (in migliaia)"
PRINT SPACE$(10); "         1         2         3"
PRINT SPACE$(10); "123456789012345678901234567890": PRINT

FOR articolo = 1 TO maxArticolo
    PRINT vendite(articolo).nomeArticolo;
    FOR bar = 1 TO vendite(articolo).vendita
        PRINT barra$(articolo);
    NEXT bar
    PRINT : PRINT
NEXT articolo
```

INSTR

Si consulti anche

LEFT$, MID$, RIGHT$

Scopo

La funzione INSTR restituisce la posizione della prima occorrenza di una determinata stringa in un'altra stringa. Questo risulta utile per la ricerca del testo nei campi di database o per la convalida dell'immissione dati da parte dell'utente.

Sintassi

```
INSTR([inizio,] stringa1,stringa2)
```

inizio è un numero intero facoltativo che indica la posizione in corrispondenza della quale viene iniziata la ricerca; se questo valore non viene specificato, la ricerca inizia dal primo carattere.

stringa1 è la stringa in cui viene condotta la ricerca, mentre **stringa2** è la stringa che la funzione **INSTR** cerca di individuare. Entrambe possono essere variabili stringa, costanti stringa, letterali di tipo stringa oppure espressioni stringa.

Utilizzo

```
PRINT INSTR("San Francesco", "San")
```

Visualizza il valore 1, dato che "San" è contenuta nella stringa "San Francesco" a partire dal primo carattere.

```
a$ = "San Francesco"
posizione% = INSTR(a$, "an")
```

Assegna alla variabile **posizione%** il valore 2, dato che la prima occorrenza di "an" ha inizio nella seconda posizione.

```
PRINT INSTR(5, "San Francesco", "an")
```

Visualizza il valore 7, dato che "an" si trova nella settima posizione quando la ricerca inizia alla posizione 5.

Descrizione

La funzione **INSTR** restituisce la posizione della prima occorrenza di una stringa in un'altra. Se si desidera unicamente verificare la presenza di una stringa all'interno di un'altra, è sufficiente utilizzare un'istruzione quale:

```
IF INSTR(a$, b$) THEN
```

Questa condizione è sempre vera se **b$** è contenuto in **a$**, dato che il valore restituito da **INSTR** è diverso da zero. Se fosse necessario conoscere l'esatta posizione in corrispondenza della quale è stata individuata la stringa, è possibile assegnarla a una variabile, nel seguente modo:

```
individuata% = INSTR(a$, b$)
```

se **INSTR** non individua la stringa, la posizione restituita è 0.

La prima stringa specificata dalla chiamata di funzione **INSTR** può essere preceduta da un numero che indica la posizione di inizio della ricerca. A questo punto è possibile eseguire un ciclo e individuare tutte le ricorrenze della stringa ricercata nel seguente modo:

```
inizio% = 1
DO
   posizione% = INSTR(inizio%, a$, b$)
   IF posizione% <> 0 THEN PRINT posizione%
   inizio% = posizione% + 1
LOOP WHILE posizione% <> 0
```

Errori

Se la posizione facoltativa di inizio è 0 o un valore inferiore, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita". Se invece è superiore rispetto alla lunghezza della stringa in cui viene effettuata la ricerca, non viene visualizzato alcun messaggio di errore, ma **INSTR** si limita a restituire 0 (nessuna corrispondenza). Se le stringhe indicate non sono una variabile stringa, una costante stringa o un letterale di tipo stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato richiede di immettere una stringa in cui effettuare una ricerca, una stringa da ricercare all'interno della prima e una stringa di sostituzione. Le funzioni LEFT$ e RIGHT$ cancellano dalla stringa qualsiasi stringa corrispondente individuata. La stringa originale viene poi ricostruita con la stringa di sostituzione al posto di quella ricercata. L'istruzione DO...LOOP ripete l'operazione di ricerca e sostituzione fino a quando la funzione INSTR non individua più alcuna occorrenza della stringa ricercata.

```
CLS
INPUT "Inserire una stringa"; unaStringa$
INPUT "Ricerca per:"; ricerca$
INPUT "Sostituisci con:"; sostituisci$
nuovaPosizione% = 1

DO WHILE nuovaPosizione% > 0
    nuovaPosizione% = INSTR(unaStringa$, ricerca$)
    IF nuovaPosizione% > 0 THEN
        temp1$ = LEFT$(unaStringa$, nuovaPosizione% - 1)
        temp2$ = RIGHT$(unaStringa$, (LEN(unaStringa$) - nuovaPosizione% -
        LEN(ricerca$) + 1))
        unaStringa$ = temp1$ + sostituisci$ + temp2$
    END IF
LOOP
PRINT unaStringa$
```

LCASE$

Si consulti anche

CHR$, UCASE$

Scopo

La funzione LCASE$ restituisce una copia della stringa indicata, in cui tutte le lettere maiuscole sono state convertite in minuscole; può essere utilizzata per effettuare confronti che devono tener conto delle lettere maiuscole o minuscole. LCASE$ può anche essere utilizzata nella prima fase di normalizzazione un testo creato completamente in lettere maiuscole.

Sintassi

```
LCASE$(stringa)
```

stringa può essere una qualsiasi variabile stringa, costante stringa, letterale di tipo stringa o espressione stringa.

Utilizzo

```
PRINT LCASE$("UTILIZZARE TUTTE LETTERE MAIUSCOLE SIGNIFICA URLARE")
```

Stampa la stringa "utilizzare tutte lettere maiuscole significa urlare".

Descrizione

La funzione LCASE$ restituisce una copia della variabile stringa, della costante stringa, del letterale di tipo stringa oppure dell'espressione stringa che sono stati specificati. Nella copia, qualsiasi lettera maiuscola della stringa originale viene convertita in minuscolo. Questa funzione non modifica i caratteri non alfabetici.

La funzione UCASE$ funziona in modo analogo, tranne per il fatto che converte tutte le lettere minuscole in maiuscole.

Errori

Se il valore fornito alla funzione LCASE$ non è una stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Specificando una stringa nulla, non viene visualizzato alcun messaggio, ma viene semplicemente restituita una stringa nulla.

Esempio

Il programma di seguito riportato accetta una stringa e converte qualsiasi carattere inserito in lettere maiuscole in minuscolo. Per uscire dal ciclo è sufficiente premere INVIO.

```
DO
    INPUT "Inserire una stringa: "; unaStringa$
    PRINT LCASE$(unaStringa$)
LOOP UNTIL unaStringa$ = ""
```

LEFT$

Si consulti anche

INSTR, MID$ (istruzione), RIGHT$

Scopo

La funzione LEFT$ restituisce una copia di una stringa consistente di un numero specificato di caratteri a partire dalla parte sinistra (dall'inizio). Utilizzando insieme LEFT$ e INSTR, è possibile estrarre la parte di una stringa fino a (o compresa) la sottostringa specificata.

Sintassi

```
LEFT$(stringa, numero)
```

stringa può essere una qualsiasi variabile stringa, costante stringa, letterale di tipo stringa o espressione stringa.

numero può essere qualsiasi numero nell'intervallo da 0 a 32767.

Utilizzo

```
a$ = "Associazione dei programmatori in QBasic"
PRINT LEFT$(a$,12)
PRINT LEFT$(a$,99)
```

Visualizza "Associazione" e "Associazione dei programmatori in QBasic".

Descrizione

La funzione **LEFT$** restituisce una stringa contenente un copia del numero di caratteri specificato, partendo dall'inizio della stringa originale. Indicando un numero di caratteri uguale o superiore alla lunghezza della stringa, **LEFT$** restituisce una copia di tutta la stringa; ad esempio l'istruzione:

```
LEFT$(a$, LEN(a$))
```

restituisce una copia esatta di **a$**.

L'esempio di seguito riportato illustra l'utilizzo di **LEFT$** con **INSTR** per restituire le parti selezionate di una stringa.

```
a$ = "Associazione dei programmatori in QBasic"
b$ = "Basic"
PRINT LEFT$(a$, INSTR(a$, b$) - 1) 'visualizza i caratteri nella sottostringa
PRINT LEFT$(a$, INSTR(a$, b$) + LEN(b$) - 1)     'visualizza i caratteri
                                                 'compresa la sottostringa
```

Il programma visualizza le stringhe "Q" e "QBasic".

Errori

Se il primo valore nella chiamata a **LEFT$** non è una stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Se viene specificata una stringa nulla o zero caratteri, non viene visualizzato alcun errore, ma **LEFT$** si limita a restituire una stringa nulla. Se viene indicato un numero inferiore a zero caratteri, QBasic visualizza il messaggio di errore: "Chiamata di funzione non valida". Se vengono indicati più di 32767 caratteri, QBasic visualizza il messaggio di errore: "Overflow".

Esempio

Il programma di seguito riportato consente di estrarre il nome di battesimo da un nome completo e di realizzare la parte introduttiva di una lettera.

```
INPUT "Inserire nome e cognome insieme: "; nomecognome$
primoSpazio% = INSTR(nomecognome$, " ")
nome$ = LEFT$(nomecognome$, primoSpazio% - 1)
PRINT "Caro "; nome$; ","
```

LEN

Si consulti anche

VARPTR, VARSEG

Scopo

La funzione **LEN** restituisce il numero di caratteri di una stringa oppure il numero di byte allocati per una variabile numerica.

Sintassi

```
LEN(var)
```

var può essere una qualsiasi variabile numerica (non un letterale numerico) qualsiasi variabile stringa, costante stringa, letterale di tipo stringa oppure espressione stringa.

Utilizzo

```
a$ + "QBasic"
PRINT LEN(a$)
```

Stampa la lunghezza della stringa a$, ovvero 6.

```
totale% = 6
PRINT LEN(totale%)
```

Stampa il numero di byte allocato per totale%, ovvero 2.

Descrizione

Per quanto riguarda le stringhe, il valore restituito da LEN è uguale al numero di caratteri presenti; per una variabile numerica, oppure per una variabile di tipo definito dall'utente, il valore restituito è il numero di byte utilizzato per memorizzare il valore della variabile.

La funzione LEN viene utilizzata soprattutto per ottenere la lunghezza di una stringa. Il valore 0 indica una stringa vuota (nulla); ne consegue che è possibile forzare l'immissione di una stringa non nulla utilizzando il seguente ciclo:

```
DO
   INPUT scelta$
LOOP UNTIL LEN(scelta$)
```

Il ciclo viene ripetuto fino a quando la lunghezza di scelta$ è un valore diverso da zero. Oppure, è possibile creare un ciclo che funziona quando l'utente immette una stringa nulla (operazione che può essere eseguita premendo INVIO).

```
DO
   INPUT "Scelta (o <Invio> per uscire)"; scelta$
   IF LEN(scelta$) = 0 THEN EXIT DO
   'elaborare la scelta
LOOP
```

La funzione LEN può anche essere utilizzata per ottenere la lunghezza di una stringa da sfruttare in un ciclo che ne elabora uno alla volta i caratteri. Questo metodo viene illustrato nel seguente ciclo FOR:

```
FOR car% = 1 TO LEN(a$)
   ca$ = MID$(a$,1,1) 'passare al carattere successivo
   'elaborare in qualche modo il carattere
NEXT car%
```

Il vantaggio principale dell'utilizzo di LEN per le variabili numeriche consiste nel fatto che questa funzione permette di determinare la lunghezza del tipo definito dall'utente, consentendo quindi di valutare il numero di byte richiesto da ogni variabile di quel tipo (si noti che, quando si tratta di un array, LEN restituisce il numero di byte per elemento).

Errori

Se alla variabile **LEN** viene fornito un valore che non è una variabile numerica né una stringa, QBasic visualizza il messaggio di errore: "È necessaria una variabile".

E importante notare che la funzione **LEN** restituisce soltanto il numero di byte necessari per memorizzare il valore effettivo di una variabile, non la quantità totale di memoria utilizzata da essa. Ogni stringa, ad esempio, ha un descrittore che utilizza altri 4 byte.

Esempio

Il programma di seguito riportato sottolinea la stringa immessa dall'utente e utilizza la funzione **LEN** per fornire il valore finale di un ciclo **FOR** che crea il carattere esteso IBM 196, utilizzato in questo caso come carattere di sottolineatura.

```
sottolineatura$ = CHR$(196)
INPUT "Inserisci una stringa che sarà sottolineata: "; unaStringa$
PRINT : PRINT unaStringa$
FOR carattere% = 1 TO LEN(unaStringa$)
    PRINT sottolineatura$;
NEXT carattere%
PRINT
```

MID$ (Funzione)

Si consulti anche

LEFT$, LEN, RIGHT$

Scopo

La funzione **MID$** restituisce una copia di una stringa. La copia inizia in una posizione determinata e comprende il numero specificato di caratteri. Questa funzione puo essere utilizzata, ad esempio, per estrarre il nome di battesimo di una persona da una stringa che contiene il nome completo.

Sintassi

```
MID$(stringa, inizio[, num])
```

stringa è una qualsiasi variabile stringa, costante stringa, letterale di tipo stringa oppure espressione stringa.

inizio è la posizione del primo carattere da estrarre.

num è il numero di caratteri da estrarre.

Utilizzo

```
PRINT MID$("QBasic: Manuale di riferimento", 2, 5)
```

Visualizza la stringa "Basic".

```
PRINT MID$("QBasic: Manuale di riferimento", 2)
```

Visualizza la stringa "Basic: Manuale di riferimento".

Descrizione

La funzione MID$ consente di estrarre una sottostringa da una stringa; è necessario specificare la stringa da cui si desidera estrarre la sottostringa, la posizione del primo carattere da copiare nella sottostringa e, facoltativamente, il numero di caratteri da inserire nella sottostringa (se quest'ultima variabile viene tralasciata, la funzione MID$ copia i caratteri dalla posizione di inizio specificata fino alla fine della stringa).

Se è necessario estrarre i caratteri dall'inizio o dalla fine della stringa, è meglio utilizzare le funzioni LEFT$ e RIGHT$. La posizione iniziale e il numero di caratteri specificati come argomenti in MID$ devono essere compresi nell'intervallo da 1 a 32767. Se la posizione iniziale ha un valore superiore rispetto alla lunghezza della stringa, MID$ restituisce una stringa nulla. Se il numero di caratteri specificato è superiore a quello dei caratteri presenti dalla posizione iniziale alla fine della stringa, la funzione restituisce tutti i caratteri, partendo dalla posizione iniziale (si noti che è possibile utilizzare la funzione LEN per ottenere la lunghezza di una stringa).

Errori

Se la prima voce indicata nella chiamata a MID$ non è una variabile stringa, un letterale di tipo stringa o una costante stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Se la posizione iniziale o il numero di caratteri è inferiore a 1, QBasic visualizza il messaggio di errore: "Chiamata di funzione non valida", mentre se il numero è superiore a 32767, QBasic visualizza il messaggio di errore: "Overflow".

Esempio

Il programma di seguito riportato esamina ogni carattere della stringa unaStringa$ per verificare se è uno spazio: in caso positivo, l'istruzione PRINT sposta la stampa alla riga successiva; in caso contrario, il programma stampa il carattere. Il risultato è che ogni parola della stringa viene stampata su una riga separata.

```
INPUT "Inserire una stringa: "; unaStringa$

FOR carattere% = 1 TO LEN(unaStringa$)
    questoCarattere$ = MID$(unaStringa$, carattere%, 1)
    IF questoCarattere$ = " " THEN
        PRINT      'vai alla linea successiva se il carattere è uno spazio
    ELSE PRINT questoCarattere$;        'stampa il carattere
    END IF
NEXT carattere%
```

MID$ (Istruzione)

Si consulti anche

LEFT$, MID$ (Funzione), RIGHT$

Scopo

L'istruzione MID$ consente di sostituire una parte di una stringa con un'altra; può essere utilizzata per sostituire a una parte di nome scritta in modo errato i caratteri corretti.

Sintassi

```
MID$(stringa, inizio[, num]) = sostituzione
```

stringa è una qualsiasi variabile stringa.
inizio è la posizione del primo carattere da sostituire.
num è il numero di caratteri sostituire.
sostituzione è la variabile stringa, la costante stringa, il letterale di tipo stringa o l'espressione stringa da utilizzare per la sostituzione.

Utilizzo

```
a$ = "Chi ha costruito il tetto?"
MID$(a$, 21, 1) = "l"
```

Stampa la stringa "Chi ha costruito il letto?"

Descrizione

L'istruzione MID$, al contrario dell'omonima funzione e di LEFT$ e RIGHT$, modifica la stringa specificata invece di restituire una copia modificata della stessa; è necessario indicare la variabile stringa da modificare, la posizione del primo carattere da sostituire e, facoltativamente, il numero di caratteri da sostituire. La chiamata deve essere seguita da un segno di uguale e dalla stringa di sostituzione, che può essere qualsiasi espressione stringa (costante stringa, variabile di stringa, letterale di tipo stringa o combinazione).

La posizione iniziale e il numero di caratteri da sostituire devono trovarsi nell'intervallo da 1 a 32767. La posizione iniziale non può essere superiore alla lunghezza della stringa di destinazione. Se viene specificato un numero di caratteri da sostituire superiore rispetto a quelli disponibili nella stringa di sostituzione, vengono utilizzati soltanto i caratteri disponibili. Se il numero di caratteri da sostituire rende la stringa modificata più lunga rispetto a quella originale, MID$ sostituisce i caratteri soltanto fino a raggiungere la lunghezza della stringa originale.

Errori

Se per la stringa da modificare viene utilizzato un letterale di tipo stringa invece della variabile stringa, QBasic visualizza il messaggio di errore: "È necessaria una variabile a stringa". Se la variabile specificata non è una variabile stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Se la posizione iniziale o il numero di caratteri è inferiore a 1, QBasic visualizza il messaggio di errore: "Chiamata di funzione non valida", mentre se il numero è superiore a 32767, QBasic visualizza il messaggio di errore: "Overflow".

Esempio

Il programma di seguito riportato utilizza la funzione MID$ per verificare ogni carattere della stringa, in modo da controllare l'eventuale occorrenza di vocali. In caso venga riscontrata la presenza di vocali, il programma utilizza l'istruzione MID$ per sostituire il carattere con un asterisco. Questo programma potrebbe essere utilizzato per sviluppare un gioco in cui si tratta di indovinare le parole.

```
INPUT "Inserire una stringa: "; unaStringa$

FOR carattere% = 1 TO LEN(unaStringa$)        'controlla ciascun carattere
    car$ = MID$(unaStringa$, carattere%, 1)  'ottiene il carattere successivo
    IF INSTR("AEIOU", UCASE$(car$)) THEN          'è una vocale ?
        MID$(unaStringa$, carattere%, 1) = "*"    'sostituisce la vocale con *
    END IF
NEXT carattere%

PRINT unaStringa$
```

RIGHT$

Si consulti anche

INSTR, LEFT$, MID$ (Istruzione)

Scopo

La funzione RIGHT$ restituisce una copia di una stringa, consistente di un numero specificato di caratteri a partire dalla parte destra (dalla fine); può essere utilizzata per estrarre l'ultima parola di una stringa.

Sintassi

```
RIGHT$(stringa, numero)
```

stringa può essere una qualsiasi variabile stringa, costante stringa, letterale di tipo stringa o espressione stringa.

numero può essere qualsiasi numero nell'intervallo da 0 a 32767.

Utilizzo

```
a$ = "tutto il mondo"
PRINT RIGHT$(a$, 5)
PRINT RIGHT$(a$, 99)
```

Stampa la stringa "mondo" e "tutto il mondo".

Descrizione

La funzione RIGHT$ restituisce una stringa contenente un copia del numero di caratteri specificato, partendo dalla fine della stringa originale. Specificando un numero di caratteri uguale o superiore alla lunghezza della stringa, RIGHT$ restituisce una copia di tutta la stringa; ad esempio l'istruzione:

```
RIGHT$(a$, LEN(a$))
```

restituisce una copia esatta di a$.

Errori

Se il primo valore nella chiamata a RIGHT$ non è una stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Se viene specificata una stringa nulla o di zero caratteri, non viene visualizzato alcun errore, ma RIGHT$ si

limita a restituire una stringa nulla. Se viene specificato un numero inferiore a zero caratteri, QBasic visualizza il messaggio di errore: “Chiamata di funzione non valida”. Se vengono indicati più di 32767 caratteri, QBasic visualizza il messaggio di errore: “Overflow”.

Esempio

Il programma di seguito riportato consente di modificare una stringa contenente un nome, in modo tale che il cognome venga elencato come la prima parola della stringa. Questo programma potrebbe essere utilizzato per formattare un elenco di nomi per poi in seguito eseguire un ordinamento. Il programma si avvale della funzione **RIGHT$** per estrarre l’ultimo nome dalla stringa.

```
INPUT "Inserire un nome completo: "; nomecognome$

lunghezza% = LEN(nomecognome$)
FOR carattere% = lunghezza% TO 1 STEP -1          'esamina ciascun carattere
    IF MID$(nomecognome$, carattere%, 1) = " " THEN     'iniziando alla fine
                                                        'del nome
        EXIT FOR                                  'trovato l'inizio del cognome
    END IF
NEXT carattere%

IF carattere% <> 0 THEN
    cognome$ = RIGHT$(nomecognome$, LEN(nomecognome$) - carattere%)
'estrae il cognome
    restoDelNome$ = LEFT$(nomecognome$, carattere% - 1)
    PRINT cognome$; ", "; restoDelNome$
ELSE
    PRINT "Devi immettere un nome intero (nome e cognome)"
END IF
```

SPACE$

Si consulti anche

LSET, PRINT USING, RSET, SPC, TAB

Scopo

La funzione **SPACE$** restituisce una stringa contenente un numero specificato di spazi. Questa funzione può risultare utile per far rientrare una parte di testo.

Sintassi

```
SPACE$(numero)
```

numero può essere qualsiasi numero nell’intervallo da 0 a 32767.

Utilizzo

```
PRINT SPACE$(10); "Testo rientrato"
```

Stampa la stringa “Testo rientrato” preceduta da 10 spazi.

Descrizione

La funzione SPACE$ restituisce una stringa consistente di un numero specificato di spazi (carattere ASCII 32). Il numero indicato deve trovarsi nell'intervallo da 0 a 32767; se non è un intero, QBasic lo arrotonda al numero intero più vicino.

Le istruzioni SPC e TAB possono essere utilizzate per ottenere un risultato analogo nella formattazione della copia su carta.

Errori

Se il valore specificato con SPACE$ non è numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Se viene utilizzato un valore numerico inferiore a 0, QBasic visualizza il messaggio di errore: "Chiamata di funzione non valida", mentre se il valore numerico è superiore a 32767, QBasic visualizza il messaggio di errore: "Overflow". Utilizzando il valore 0, non viene prodotto alcun messaggio, ma SPACE$ si limita a restituire una stringa nulla.

Esempio

Il programma di seguito riportato definisce la funzione Rientro$, che accetta un valore di rientro e una stringa e utilizza la funzione SPACE$ per far rientrare la stringa quando viene visualizzata. Il programma si avvale di Rientro$ per formattare le informazioni presenti su una pagina.

```
DECLARE FUNCTION Rientro$ (spazi!, unaStringa$)

CLS
PRINT Rientro$(25, "Informazione personale");
PRINT Rientro$(15, "Pagina 1"): PRINT
PRINT Rientro$(25, "Identificazione"): PRINT
PRINT Rientro$(5, "Nome  :"): PRINT
PRINT Rientro$(5, "Via   :"): PRINT
PRINT Rientro$(5, "Città :");
PRINT Rientro$(30, "Stato :");
PRINT Rientro$(10, "CAP   :")

FUNCTION Rientro$ (spazi, unaStringa$)
  Rientro$ = SPACE$(spazi) + unaStringa$   'restituisce una stringa rientrata
END FUNCTION
```

STR$

Si consulti anche

VAL

Scopo

La funzione STR$ restituisce la rappresentazione di un valore come stringa, consentendo in questo modo di gestire un numero come stringa e di applicare a esso le altre funzioni di gestione delle stringhe per la convalida e la formattazione.

Sintassi

```
STR$(numero)
```

numero può essere una qualsiasi variabile numerica, costante numerica, letterale numerico o espressione numerica.

Utilizzo

```
num! = 3.1415
PRINT STR$(num!)
```

Stampa la stringa "3.1415".

Descrizione

La funzione STR$ accetta qualsiasi espressione numerica corretta e restituisce il risultato sotto forma di stringa. La funzione VAL esegue l'operazione opposta, ovvero accetta una stringa e restituisce il numero rappresentato dal carattere.

Errori

Se il valore specificato con STR$ non è numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Se viene utilizzato un valore che inizia con dei numeri, ma contiene caratteri che non sono validi in un numero o un'espressione (ad esempio una virgola), QBasic visualizza il messaggio di errore: "Era atteso:" ed evidenzia il carattere errato.

Esempio

L'esempio di seguito riportato illustra la modalità da seguire per stampare un numero senza spazi iniziali o finali. Ad esempio, quando viene stampato l'importo su un assegno è necessario eliminare qualsiasi spazio iniziale e finale che potrebbe essere utilizzato per modificare la cifra. La prima istruzione PRINT, che visualizza una variabile numerica, comprende uno spazio iniziale e uno finale che devono essere eliminati; la seconda istruzione PRINT non visualizza tali spazi, dato che l'importo numerico è stato convertito in una stringa con la funzione STR$ e quindi gli spazi eliminati con le funzioni LTRIM$ e RTRIM$.

```
CLS
INPUT "Quanti soldi vuoi"; numero#
PRINT "***"; numero#; "***"; " lire"

numero$ = STR$(numero#)                'converte un numero in una stringa
numero$ = LTRIM$(RTRIM$(numero$))      'elimina gli spazi iniziali e finali

PRINT "***"; numero$; "***"; " lire"
```

STRING$

Si consulti anche

CHR$, SPACE$

Scopo

La funzione STRING$ restituisce una stringa contenente un carattere che viene ripetuto un numero specificato di volte. Questa funzione può essere utilizzata per creare sottolineature, righe composte da asterischi, finestre con bordi e così via.

Sintassi

```
STRING$(numero, codice)
```

oppure

```
STRING$(numero, stringa)
```

numero è il numero di caratteri (compreso nell'intervallo da 0 a 32767) della stringa risultante.

codice è il codice del carattere ASCII o del carattere esteso IBM per i caratteri desiderati nella stringa restituita.

stringa è una qualsiasi variabile stringa, costante stringa, letterale di tipo stringa o espressione stringa.

Utilizzo

```
PRINT STRING$(10, 42)
```

Stampa la stringa "**********"

```
PRINT STRING$(10, "*")
```

Stampa la stringa "**********".

```
a$ = STRING$(20,"Basic")
```

Assegna la stringa "BBBBBBBBBBBBBBBBBBB" alla variabile a$.

Descrizione

La funzione STRING$ risulta utile per la creazione di una stringa che comprende caratteri ripetuti; il primo argomento indica il numero di caratteri presenti nella stringa, il secondo è un codice di carattere oppure una variabile stringa, una costante stringa, un letterale di tipo stringa o un'espressione stringa e indica il carattere ripetuto dalla funzione stessa. Se il secondo argomento è una stringa contenente più di un carattere, viene utilizzato soltanto il primo.

Un letterale di tipo stringa deve essere utilizzato nella chiamata a STRING$, soltanto se può essere immesso facilmente da tastiera. In caso contrario, ad esempio per i caratteri di controllo o quelli estesi IBM, è meglio utilizzare il relativo codice.

Se viene utilizzato un codice di carattere con un valore superiore a 255, ad esempio 256, esso produce lo stesso carattere creato da 0 e così via. I codici negativi si comportano in modo analogo, ovvero il valore -1 produce lo stesso carattere di 255.

Se si desidera creare una stringa che comprende soltanto spazi è meglio utilizzare la funzione SPACE$.

Errori

Se il numero specificato per quanto riguarda il numero di caratteri è negativo, QBasic visualizza il messaggio di errore: "Chiamata di funzione non valida". Se il valore numerico è superiore a 32767, QBasic visualizza il messaggio di errore: "Overflow". Se non viene indicato alcun carattere il risultato è una stringa nulla,

mentre se viene specificata una stringa come primo argomento, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Se viene utilizzato un codice di carattere inferiore a -32768 o superiore a 32767, QBasic visualizza il messaggio di errore: "Overflow".

Esempio

Il sottoprogramma **RiquadraStringa**, presente nel programma di seguito riportato, consente di definire caratteri grafici e quindi di utilizzare **CHR$** (per i caratteri singoli) e **STRING$** (per più caratteri) per tracciare un riquadro attorno a una stringa. Il sottoprogramma verifica la posizione richiesta del riquadro e stampa un messaggio se tutto il riquadro non può essere inserito nello schermo standard.

```
DECLARE SUB RiquadraStringa (riga!, colonna!, unaStringa$)

CLS
messaggio$ = "Guida di riferimento a QBasic"
RiquadraStringa 10, 30, messaggio$

SUB RiquadraStringa (riga, colonna, unaStringa$)  'stampa una stringa in un
                                                  'riquadro
    'caratteri grafici
    CONST SUPSINISTRO = 201
    CONST INFSINISTRO = 200
    CONST SUPDESTRO = 187
    CONST INFDESTRO = 188
    CONST ORIZZONTALE = 205
    CONST VERTICALE = 186

    CONST MAXLINEE = 22          'ultima linea a cui il riquadro può iniziare
    CONST MAXCOLONNE = 80        'massimo numero di colonne del video
    IF riga > MAXLINEE THEN
       PRINT "Una riga deve essere di 22 caratteri o meno."
       EXIT SUB
    END IF
    piuASinistra = MAXCOLONNE - LEN(unaStringa$) - 2

    IF colonna > piuASinistra THEN                 'oltre il margine sinistro
       PRINT "La riga per la stringa "; unaStringa$; " non deve essere
       maggiore di ";
       PRINT piuASinistra
       EXIT SUB
    END IF
    LOCATE riga, colonna
    linea$ = STRING$(LEN(unaStringa$), ORIZZONTALE)

    LOCATE riga + 1, colonna
    PRINT CHR$(SUPSINISTRO); linea$; CHR$(SUPDESTRO)  'stampa la parte
                                                       'superiore del riquadro

    LOCATE riga + 2, colonna
    PRINT CHR$(VERTICALE); unaStringa$; CHR$(VERTICALE) 'stampa i lati del
                                                         'riquadro

    LOCATE riga + 3, colonna
    PRINT CHR$(INFSINISTRO); linea$; CHR$(INFDESTRO)  'stampa la parte
                                                       'inferiore del riquadro
END SUB
```

UCASE$

Si consulti anche

LCASE$

Scopo

La funzione UCASE$ restituisce una copia della stringa indicata, in cui tutte le lettere minuscole sono convertite in maiuscole. Questa funzione può essere utilizzata per controllare immissioni da parte di un utente in caratteri maiuscoli o minuscoli.

Sintassi

```
UCASE$(stringa)
```

stringa può essere una qualsiasi variabile stringa, costante stringa, letterale di tipo stringa o espressione stringa.

Utilizzo

```
a$ = "Qbasic"
PRINT UCASE$(a$)
```

Stampa la stringa "QBASIC".

Descrizione

La funzione UCASE$ restituisce una copia della variabile stringa, della costante stringa, del letterale di tipo stringa oppure dell'espressione stringa specificati, in cui qualsiasi lettera in minuscolo viene convertita in maiuscolo. Questa funzione non modifica le lettere già in maiuscolo o i caratteri non alfabetici.

La funzione LCASE$ opera in modo analogo, tranne per il fatto che converte tutte le lettere maiuscole in minuscole.

Utilizzando le funzioni LEFT$ e RIGHT$ con l'istruzione MID$, è possibile sfruttare UCASE$ per trasformare in lettere maiuscole soltanto alcuni caratteri di una stringa. Ad esempio, la parte di programma di seguito riportata stampa la stringa "Maria".

```
a$ = "maria"
MID$(a$, 1, 1) = UCASE$(LEFT$(a$, 1))
PRINT a$
```

Errori

Se il valore fornito alla funzione UCASE$ non è una stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Se viene invece specificata una stringa nulla, non viene visualizzato alcun messaggio, ma viene semplicemente restituita una stringa nulla.

Esempio

Il programma di seguito riportato richiede all'utente di immettere le lettere A, B, C, D o U. La funzione UCASE$ converte qualsiasi lettera minuscola in maiuscola, in modo che il programma debba verificare soltanto le lettere maiuscole nelle istruzioni

IF che gestiscono l'immissione. Il programma si avvale di LEFT$ per eliminare tutti i caratteri di immissione presenti dopo il primo, permettendo in questo modo agli utenti di inserire parole complete (ad esempio "copia").

```
CLS
PRINT "<Inserire U per uscire dal programma>"
DO
  INPUT "Inserire A, B, C, D, o U"; scelta$
  scelta$ = UCASE$(LEFT$(scelta$, 1))
  IF INSTR("ABCDU", scelta$) THEN
      PRINT "Hai scelto: "; scelta$
  END IF
LOOP UNTIL scelta$ = "U"
```

VAL

Si consulti anche

STR$

Scopo

La funzione VAL restituisce il numero rappresentato dalle cifre (e da qualsiasi altro simbolo numerico) contenute in una variabile stringa o espressione stringa. Questa funzione consente di accettare le immissioni numeriche sotto forma di stringa, di utilizzare le diverse funzioni di stringa per convalidare i dati immessi e quindi di convertirli nuovamente in un numero da utilizzare nei calcoli.

Sintassi

```
VAL(stringa)
```

stringa può essere una qualsiasi variabile stringa, costante stringa, letterale di tipo stringa o espressione stringa.

Utilizzo

```
INPUT num$
valore = VAL(num$)
```

Assegna alla variabile valore il valore numerico della stringa immessa.

Descrizione

La funzione VAL accetta il valore di una variabile stringa e la esamina, partendo dal primo carattere, fino a quando non viene individuato un carattere che non può far parte di un letterale numerico (ad esempio una virgola, una lettera o il simbolo di dollaro), oppure finché non viene raggiunta la fine della stringa. La stringa risultante (che può comprendere cifre, un segno più o meno, un punto decimale oppure l'esponenziale D o E) viene quindi convertita in un valore numerico valido. Le stringhe di seguito riportate rappresentano un esempio.

Valore della stringa	Risultato della chiamata a VAL
500	500
4.4E7	44000000
123D-7	.0000123
99999999999999999	1D+17
4,000	4
xvi	0
2b or not 2b	2

Si noti che è possibile immettere i numeri utilizzando la notazione esponenziale (con D o E). VAL converte i numeri in una normale notazione decimale, a patto che non abbiano più di 16 cifre; in tal caso rimangono in notazione esponenziale. I numeri con più di 16 cifre immesse nella notazione decimale vengono convertiti in notazione esponenziale.

La funzione STR$ esegue l'operazione contraria, dato che restituisce una stringa contenente la rappresentazione di un numero.

Commenti

VAL non consente di verificare gli errori. Spetta al programma creato controllare l'immissione di stringhe errate, quali "4,000", "50%", "1/2" e così via, e rispondere in ogni caso in modo corretto.

Errori

Se l'espressione specificata con VAL non è una stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato accetta l'immissione dei dati, li memorizza in una variabile stringa e utilizza VAL per convertirla in un numero, che viene memorizzato in un array. Sia i dati immessi che il valore convertito vengono visualizzati dopo ogni immissione. Il programma controlla la lunghezza della stringa immessa mediante la funzione LEN se è 0 (ovvero se è stato premuto INVIO), il programma interrompe la richiesta di immissione.

```
DIM numeri(10)          'array per gestire i numeri inseriti
CLS
FOR i% = 1 TO 10
    INPUT "Inserire un numero, premere Invio per uscire:", numero$  'valore
                                                                    'inserito
                                                                    'come
                                                                    'stringa
    IF LEN(numero$) = 0 THEN              'l'utente ha premuto Invio
        i% = i% - 1                       'non conteggiare questa iterazione
        EXIT FOR                          'uscita dal ciclo
    END IF

    valore = VAL(numero$)                 'converte l'input in un numero
    PRINT "===> "; numero$, valore        'visualizza stringa e valore
    numeri(i%) = valore                   'memorizza il numero nell'array
```

```
NEXT i%

totale = 0
FOR j% = 1 TO i%
    totale = totale + numeri(j%)        'addiziona i valori dell'array
NEXT j%

IF i% = 0 THEN PRINT "Nessun numero inserito": SYSTEM  'se non viene inserito
                                                       'alcun numero, uscita

media = totale / i%                    'trova la media
PRINT "*** Totale: "; totale; "*** Media: "; media
```

Capitolo 6

Array e dati

INTRODUZIONE

Gli array rappresentano gli strumenti più efficaci per la memorizzazione e il caricamento in memoria di grandi quantità di dati, esattamente come i file sono strumenti validi per la memorizzazione e il caricamento di dati su disco. Alcuni dei dati utilizzati dai programmi sono legati a determinati valori, ad esempio la variabile numerica contenente il totale delle vendite mensili oppure la variabile stringa contenente il nome dell'utente. In molti casi, tuttavia, un programma deve gestire molte voci contenenti dati dello stesso tipo e di scopo analogo. Si supponga ad esempio di dover elaborare i totali delle vendite relativi ai 12 mesi dell'anno e di dover stampare 12 stringhe, ognuna delle quali contiene il nome di un mese; in questo caso, tutte le voci dei dati hanno lo stesso tipo (a precisione semplice e stringa) e scopo (i totali delle vendite e il nome del mese).

Sarebbe ovviamente noioso creare variabili quali TotaleGen, TotaleFeb, TotaleMar e così via per i totali delle vendite e gen$, feb$, mar$ per i nomi dei mesi; non soltanto tutti i nomi dovrebbero essere inseriti nel programma, ma non sarebbe facile fare riferimento a ogni totale o nome di volta in volta. Si supponga di voler aggiungere tutti i totali mensili per creare un unico totale: sarebbe necessario scrivere un'istruzione quale TotaleGlobale = TotaleGen + TotaleFeb + TotaleMar ... + TotaleDec. Oppure, si supponga di trovarsi nella fase di stampa dei prospetti relativi a tutto l'anno e di voler stampare anche il nome del mese in corso; in questo caso sarebbe necessario fare riferimento alla variabile stringa corretta per ogni mese, ad esempio, se il mese è luglio si dovrebbe utilizzare la variabile luglio$ e così via.

In questo capitolo viene affrontata la modalità di utilizzo degli array come strumenti atti alla organizzazione e all'elaborazione dei dati. Un array consente di creare un insieme di variabili con un nome comune, ad esempio totali; in effetti, fornisce un casellario con molti cassetti, numerati da 0 (o talvolta 1) a qualsiasi numero indicato. Il contenuto di un determinato cassetto può essere caricato (o memorizzato) semplicemente facendo riferimento al nome dell'array e al numero del cassetto. Se

invece si desidera elaborare in modo sistematico il contenuto di tutti i cassetti, è sufficiente utilizzare un semplice ciclo (ad esempio il ciclo FOR) per accedere a ogni cassetto, uno alla volta.

Nella Tabella 6.1 sono riportate in ordine alfabetico le istruzioni e le funzioni relative agli array e ai dati.

Tabella 6.1 Le istruzioni e le funzioni relative agli array e ai dati

Istruzione o funzione	Scopo
DATA	Specifica una serie di valori di dati (generalmente da leggere in array)
DIM	Dimensiona (ovvero dichiara e alloca la memoria) un array oppure indica il tipo di dati di una variabile
$DYNAMIC	Rende dinamici tutti gli array numerici e gli array stringa a lunghezza fissa
ERASE	Reinizializza un array statico o libera la memoria utilizzata da un array dinamico
LBOUND	Restituisce il limite inferiore (il più piccolo indice disponibile) della dimensione indicata di un array
OPTION BASE	Imposta 0 o 1 come indice inferiore di default
READ	Assegna alla variabile indicata la voce successiva in un'istruzione DATA
REDIM	Ridefinisce gli intervalli degli indici per un array
RESTORE	Indica l'istruzione DATA successiva da utilizzare per la lettura dei dati
$STATIC	Rende tutti gli array statici
UBOUND	Restituisce il limite superiore (l'indice più grande disponibile) della dimensione indicata di un array

DICHIARAZIONE E UTILIZZO DEGLI ARRAY

L'istruzione DIM (dimensione) consente di dichiarare il nome, le dimensioni e il tipo di un array. Nella maggior parte dei casi è sufficiente utilizzare la parola chiave DIM seguita dal nome dell'array e dalle dimensioni, come negli esempi seguenti.

```
DIM totali!(12)
DIM mesi$(12)
```

Queste istruzioni dichiarano due array, quello denominato totali! può contenere il totale delle vendite di ogni mese, mentre il secondo, denominato mesi$, può comprendere il nome di ogni mese. Si noti che i nomi degli array possono utilizzare lo stesso carattere di dichiarazione del tipo che indica i tipi di dati delle normali variabili. In questo esempio, totali! è un array di variabili numeriche a precisione

singola, mentre mesi$ è un array di stringhe. È possibile inoltre utilizzare la clausola AS per specificare il tipo di dati.

```
DIM totali(12) AS SINGLE
DIM mesi(12) AS STRING
```

Si noti che non è possibile utilizzare contemporaneamente i suffissi del tipo e la clausola AS, ma soltanto uno dei due. Per applicare un tipo ai nomi degli array è anche possibile sfruttare l'istruzione DEFtipo (ad esempio DEFINT o DEFSTR); nelle due istruzioni:

```
DEFSTR A-C
DIM a(5)
```

l'array a è un array di stringhe, dato che l'istruzione DEFSTR rende stringa il tipo di default per le variabili il cui nome inizia con A, B o C.

QBasic consente inoltre di dimensionare implicitamente array che hanno 10 elementi, senza dover utilizzare un'istruzione DIM. Quindi, se il programma utilizza l'istruzione:

```
grado(5) = 99
```

QBasic si comporta come se contenesse l'istruzione:

```
DIM grado(10)
```

Tuttavia, si consiglia di non utilizzare array dimensionati implicitamente, dato che il programma risulta meno leggibile.

Dimensioni e indici

Il numero o i numeri tra le parentesi che seguono il nome dell'array nell'istruzione DIM indicano il numero delle variabili individuali (denominate *elementi*) dell'array e i numeri (denominati *indici*) che possono essere utilizzati per accedere agli elementi. Le ragioni per indicare a QBasic il numero degli elementi presenti in un array sono due: innanzitutto, il numero degli elementi moltiplicato per le dimensioni di ogni elemento indica la quantità di memoria che il compilatore alloca, o riserva, per l'array; in secondo luogo, QBasic utilizza i valori specificati per verificare i riferimenti all'array in modo da impedire l'accesso a elementi non esistenti.

Nell'esempio precedente, sia l'array totali! che quello mesi$ utilizzano il valore 12 per specificare che vi sono 12 elementi in ogni array. Per default, l'indice del primo elemento (la *base*) di ogni array è il numero 0, quindi in realtà questi array hanno 13 elementi, con gli indici nell'intervallo da 0 a 12. Dato che gennaio viene generalmente considerato il primo mese e non quello numero zero, è più vantaggioso rendere 1 la base dell'array, utilizzando l'istruzione:

```
OPTION BASE 1
```

prima di dimensionare gli array. In questo modo, è possibile modificare il valore di default in modo che l'indice del primo elemento di ogni array sia il numero 1.

Array multimensionali

Talvolta è utile avere più di una dimensione in un array. Ad esempio, si supponga di dover memorizzare le informazioni riguardanti la posizione dei vari pezzi su una scacchiera: quale pezzo eventualmente si trova sul riquadro, il valore relativo di quel riquadro per il giocatore e così via; dato che una scacchiera è composta da 8 riquadri per 8, è logico dimensionare il seguente array:

```
DIM scacchiera%(8, 8)
```

L'array scacchiera% ha due dimensioni, ognuna contenente otto elementi (questo presuppone che la base sia stata impostata a 1 mediante l'istruzione OPTION BASE) e quindi ha 8 volte 8, ovvero 64, elementi. In linea generale, è possibile considerare un array a due dimensioni come una tabella o un foglio elettronico, che dispone di un determinato numero di righe e di colonne.

QBasic consente di specificare molte dimensioni in un array, anche se normalmente non se ne utilizzano più di tre. Si supponga di volere una mappa topografica in cui ogni punto indica la coordinata est-ovest, quella nord-sud e l'altezza sul livello del mare. Se la mappa misura 20 chilometri da est a ovest, 20 chilometri da nord a sud e 30 metri sopra il livello del mare, è possibile dichiarare il seguente array:

```
DIM topo(20, 20, 30)
```

Tutto ciò può essere considerato come una pila di 30 griglie 20 per 20 (oppure come una grossa scacchiera tridimensionale).

Indicazione degli intervalli degli indici

Un indice è un numero che specifica la posizione di un elemento entro la dimensione di un determinato array. Ad esempio, per accedere al nome "Febbraio", è sufficiente specificare mesi$(2), dato che "Febbraio" viene memorizzato nel secondo elemento dell'array. Per accedere al riquadro nella seconda riga della quarta colonna sulla scacchiera, è necessario utilizzare l'elemento di array scacchiera%(2, 4), come riportato nella Figura 6.1. Per accedere a un paese che si trova 8 chilometri a est, 7 a nord e ad un'altitudine di 3000 metri rispetto al punto di riferimento sulla mappa, è sufficiente utilizzare l'elemento topo(8, 7, 3). Questi esempi dimostrano che per accedere a un elemento di un array è sufficiente fornirne la posizione e le dimensioni.

Negli esempi finora esaminati si è partiti dal presupposto che ogni dimensione inizi a 0 o a 1 (in base all'impostazione dell'istruzione OPTION BASE). Sebbene questo risulti corretto nella maggior parte dei casi, talvolta è necessario indicare un indice di base diverso da 0 o da 1. Ad esempio, il sistema di coordinate cartesiane utilizza il punto specificato dalle coordinate (0, 0) come proprio centro (origine), inserisce le coordinate x negative a sinistra rispetto all'asse y e le coordinate y negative sotto l'asse x. L'array che rappresenta questo sistema di coordinate deve consentire l'utilizzo di indici negativi, che possono essere indicati dalla parola chiave TO quando vengono specificate le dimensioni dell'array con l'istruzione DIM, come nell'esempio di seguito riportato.

```
DIM cartesiano(-100 TO 100, -100 TO 100)
```

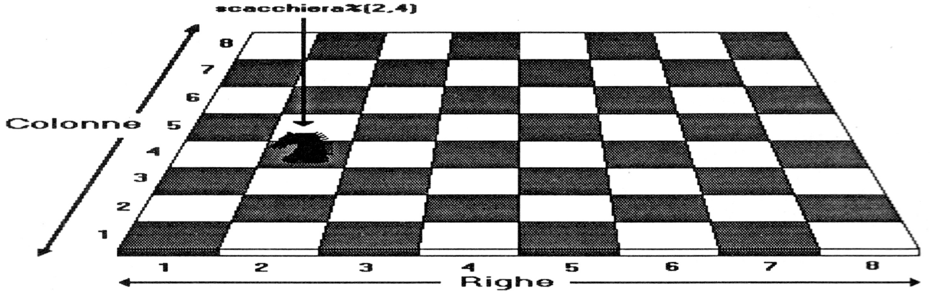

Figura 6.1 L'accesso a un elemento di un array

L'istruzione dichiara un array a due dimensioni, denominato cartesiano, i cui indici si trovano nell'intervallo da -100 a 100. Il punto indicato da (-50, -25) può essere specificato con cartesiano(-50, -25), come riportato nella Figura 6.2.

La parola chiave TO può essere utilizzata per specificare qualsiasi intervallo di indici. Si supponga di voler registrare il numero di studenti che ottiene un punteggio effettuando un test che ha un punteggio minimo di 40 e uno massimo di 100. L'istruzione di seguito riportata crea l'array totali%, che può contenere tale registrazione.

```
DIM totali%(40 TO 100)
```

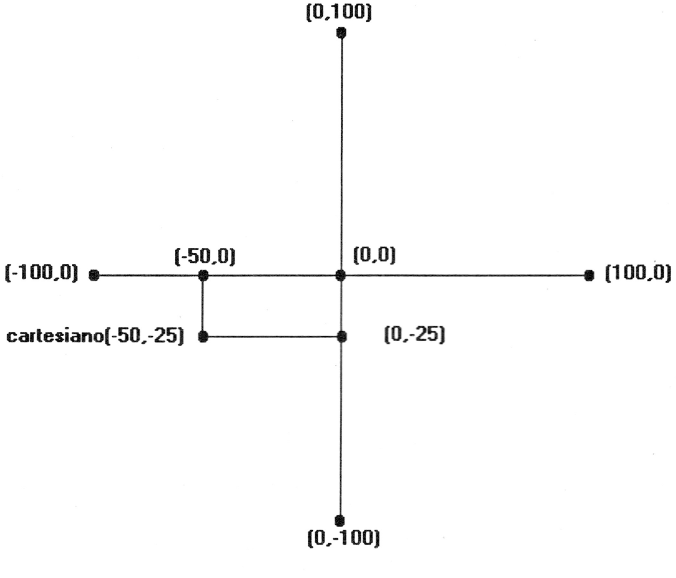

Figura 6.2 L'indicazione di un punto nel sistema di coordinate cartesiano

Memorizzazione dei dati in un array

Dopo aver creato un array, è possibile assegnare ai singoli elementi dei valori specificando il nome dell'array e l'indice dell'elemento, come riportato nelle seguenti istruzioni:

```
totali!(1) = 244.45
totali!(2) = 389.90
```

I dati relativi a un array potrebbero anche essere tratti dal programma. Ad esempio, se i dati si trovano in un file denominato VENDITE.DAT, è possibile trasferirli in un array un valore alla volta, come illustrato nel segmento di programma di seguito riportato.

```
OPEN "VENDITE.DAT" FOR INPUT AS #1
FOR totale% = 1 TO 12
   INPUT #1, totali!(totale%)
NEXT totale%
CLOSE #1
```

Si osservi che, dato che gli array organizzano i dati in modo sequenziale, in base all'ordine degli indici, è sufficiente creare un ciclo di tipo FOR per inserire i dati.

È anche possibile utilizzare l'istruzione READ per assegnare i dati agli elementi di un array. Questa istruzione viene generalmente utilizzata se le voci dei dati hanno valori fissi che sono identici ogniqualvolta il programma viene eseguito. Di seguito è riportato un esempio di come sia possibile leggere i nomi dei mesi nell'array mesi$.

```
FOR mese% = 1 TO 12
   READ mesi$(mese%)
NEXT mese%

DATA Gennaio, Febbraio, Marzo, Aprile
DATA Maggio, Giugno, Luglio, Agosto
DATA Settembre, Ottobre, Novembre, Dicembre
```

In questo esempio, ogni volta che viene eseguita l'istruzione READ a un elemento di mesi$ viene assegnata la stringa successiva dalle istruzioni DATA.

Accesso ai dati in un array

Come è già stato illustrato, è possibile accedere a un elemento di un array indicando l'indice di ciascuna dimensione. Quindi, l'istruzione:

```
PRINT mesi$(4)
```

visualizza "Aprile", mentre l'istruzione:

```
Meseattuale! = totali!(8)
```

assegna il valore memorizzato nell'ottavo elemento di totali! alla variabile Meseattuale!. Se un array ha più di una dimensione, è necessario indicare un indice per ognuna; l'istruzione:

```
valQuadrato% = scacchiera%(3, 7)
```

fa riferimento al settimo elemento nella terza serie (ovvero, nella riga 3, colonna 7 sulla scacchiera) e ne assegna il valore alla variabile valQuadrato%.

Dopo aver dichiarato un array con l'istruzione DIM, è possibile assegnare una variabile o una costante a qualsiasi elemento, oppure assegnare un valore di un elemento a qualsiasi variabile in grado di accettare quel tipo di dati. QBasic considera ogni elemento di un array come una variabile separata del tipo di dati dichiarato per l'array.

Array di tipo definito dall'utente

Uno degli svantaggi legati all'utilizzo degli array è rappresentato dal fatto che ogni array può contenere soltanto un tipo di dati; non è infatti possibile memorizzare sia le stringhe che gli interi in un unico array. Una soluzione consiste nel creare un array per ogni tipo di dati. Se, ad esempio, si desidera utilizzare gli array per memorizzare i nomi dei dipendenti, i numeri di identificazione, gli incarichi e gli stipendi, è necessario utilizzare un array per ognuno dei quattro tipi di informazioni. Per fare riferimento ai dati relativi a un determinato dipendente, è possibile utilizzare un ciclo FOR:

```
FOR dipendente% = 1 TO totaleDipendenti%
   PRINT nomi$(dipendente%)
   PRINT id%(dipendente%)
   PRINT incarico$(dipendente%)
   PRINT stipendio(dipendente%)
NEXT dipendente%
```

Anche se questo metodo funziona, rende difficile comprendere il fatto che si stanno gestendo una serie di record di dati, ognuno dei quali descrive alcune informazioni riguardanti un particolare dipendente. Una soluzione migliore è quella di dichiarare un tipo definito dall'utente mediante l'istruzione TYPE...END TYPE e quindi utilizzare un array di record di quel tipo, come nel seguente esempio:

```
Type rec Dipendente
     dipNome AS STRING
     id AS DOUBLE
     incarico AS STRING +32
     stipendio AS SINGLE
END TYPE
DIM  dipinfo (1000) AS rec Dipendente
FOR  dipendente%=1 TO totali dipendenti
     PRINT dipinfo (dipendente%). dipNome
     PRINT dipinfo (dipendente%).id
     PRINT dipinfo (dipendente%). incarico
     PRINT dipinfo (dipendente%). stipendio
NEXT dipendente%
```

In questo caso, la creazione di un array di record consente di assegnare a un elemento dell'array le informazioni relative ad ogni dipendente. Si noti tuttavia che, anche se è possibile creare un array di variabili record, queste ultime non possono contenere un array.

Utilizzo degli array nei sottoprogrammi e nelle funzioni

Generalmente, un array viene dimensionato nel codice a livello del modulo, a meno che non debba essere utilizzato soltanto in un sottoprogramma o in una funzione particolari. Per rendere un array disponibile a un sottoprogramma o a una funzione, è possibile avvalersi di diversi metodi. Nel programma principale, l'istruzione DIM può comprendere la parola chiave SHARED, come nell'istruzione di seguito riportata.

```
DIM SHARED Datiglobali(100)
```

L'utilizzo di SHARED consente a tutti i sottoprogrammi e a tutte le funzioni di accedere direttamente a un array (invece di passarlo come parametro); in altri termini, l'array diventa globale.

È anche possibile utilizzare l'istruzione SHARED in un sottoprogramma o in una funzione particolare per consentire di sfruttare un array dichiarato nel programma principale. Infine, un array può essere passato come parametro a un sottoprogramma o a una funzione semplicemente inserendone il nome seguito da una coppia di parentesi vuote nell'elenco degli argomenti, nel seguente modo:

```
Totaleincrociato totali()
```

In questo caso, il sottoprogramma Totaleincrociato dispone anche di un array specificato (con una coppia di parentesi vuote) nel relativo elenco dei parametri.

ARRAY STATICI E DINAMICI

Se un array viene dichiarato utilizzando un letterale o una costante come dimensione, come nelle istruzioni seguenti, è statico.

```
CONST ELEMENTS = 100
DIM n(ELEMENTS)
DIM p(10)
```

Sia n che p sono array statici, QBasic conosce con esattezza le dimensioni che debbono avere e il compilatore alloca lo spazio corretto in memoria per ogni array, quando individua l'istruzione DIM a esso relativa. Gli array statici hanno però uno svantaggio: non è possibile modificarne le dimensioni in un momento successivo.

D'altro canto, se viene utilizzata una variabile per la dimensione, oppure l'array viene dichiarato in un'istruzione COMMON, prima di dimensionarlo, esso è dinamico; questo tipo di array offre il vantaggio che la memoria a esso relativa non viene allocata fino al momento dell'esecuzione. Nell'esempio di seguito riportato, gli array n e griglia sono dinamici.

```
COMMON n()
DIM n(500)

righe% = 100
Col% = 50
DIM griglia(righe%, col%)
```

Inoltre tutti gli array dichiarati all'interno di un sottoprogramma o di una funzione sono dinamici, a meno che non vengano dichiarati in un'istruzione STATIC. Dato che la maggior parte dei sottoprogrammi e delle funzioni non devono conservare i valori delle variabili tra le chiamate, l'utilizzo degli array dinamici garantisce che sia sempre disponibile la quantità massima di memoria. Questa possibilità diventa sempre più importante, mano a mano che il programma aumenta di dimensione.

La maggior parte degli array in QBasic vengono memorizzati nel *far heap*. Gli array dimensionati in modo implicito (array che contengono 10 elementi utilizzati senza averli prima dichiarati nell'istruzione DIM) e gli array dinamici di stringhe a lunghezza variabile vengono sempre memorizzati nel segmento dati di default.

I metacomandi $STATIC e $DYNAMIC possono essere utilizzati per impostare l'allocazione statica o dinamica di default per i diversi tipi di array. In base al metacomando prescelto, QBasic alloca tutti gli array in modo statico o dinamico (le stringhe a lunghezza variabile vengono ancora memorizzate nel segmento dati di default, mentre gli array dimensionati implicitamente, che sono sempre statici, non vengono influenzati da questa operazione).

Dopo aver creato un array dinamico, è possibile utilizzare l'istruzione ERASE per liberare la memoria in modo che possa essere diversamente sfruttata, oppure è possibile utilizzare l'istruzione REDIM per modificare gli intervalli dell'indice dell'array (ma non il numero delle dimensioni). Nel caso di un array statico, ERASE ne reinizializza gli elementi a 0 (oppure a una stringa nulla per gli elementi stringa), ma non libera la memoria. L'istruzione REDIM non può essere utilizzata con gli array statici.

Bibliografia

Prata, Stephen, with Harry Henderson. *The Waite Group's Microsoft QuickBASIC Primer Plus.* Redmond, Wash: Microsoft Press, 1990.

DATA

Si consulti anche

CONST, READ, RESTORE

Scopo

Le istruzioni DATA consentono di memorizzare i valori costanti che possono essere letti nelle variabili dall'istruzione READ.

Sintassi

```
DATA val1[, val2...]
```

val1 e val2 sono letterali numerici o valori stringa. Se sono presenti più valori, devono essere separati da una virgola, mentre i letterali di tipo stringa devono essere racchiusi tra doppi apici soltanto se la stringa contiene virgole, due punti oppure spazi iniziali o finali.

Utilizzo

```
DATA Nord, Sud, Est, Ovest
```

Questa istruzione memorizza quattro costanti stringa che possono essere lette soltanto in variabili stringa.

```
DATA 640, 320, 200, 16
```

Questa istruzione memorizza quattro costanti numeriche che possono essere lette in variabili numeriche o stringa.

Descrizione

Ogni istruzione DATA comprende un elenco di valori separati da virgole. Se un programma presenta una serie di istruzioni DATA, i valori contenuti in tutte le istruzioni vengono considerati come un elenco di dati. La prima istruzione READ eseguita dal programma legge il primo valore dall'elenco dei dati nella variabile specificata, la seconda istruzione READ trasferisce il secondo valore e così via. QBasic contrassegna sempre la posizione con un puntatore in modo da tenere traccia del valore successivo da leggere (il puntatore può essere ripristinato oppure impostato nel punto iniziale di un'istruzione DATA etichettata, mediante l'istruzione RESTORE).

Ogni valore dei dati viene interpretato in base al tipo di variabile specificata nell'istruzione READ: se questa è numerica, il valore di dati corrispondente deve comprendere soltanto caratteri numerici validi; se la variabile è di tipo stringa, il valore dei dati può essere di qualsiasi tipo (per utilizzare una virgola in un valore dati stringa o per mantenere gli spazi iniziali e finali, è necessario racchiudere la stringa tra apici). Un valore che consiste soltanto di una virgola viene considerato come valore nullo e quando viene letto da un'istruzione READ, quast'ultima imposta le variabili numeriche a 0 e le variabili stringa a una stringa nulla (vuota).

QBasic consente l'utilizzo delle istruzioni DATA soltanto nel programma principale, mentre non è possibile inserirle all'interno dei sottoprogrammi o delle funzioni; invece le istruzioni READ possono essere immesse in un punto qualsiasi all'interno del programma.

Le istruzioni DATA vengono generalmente utilizzate per mantenere una serie di valori che vengono letti da un ciclo in un array. Tuttavia, l'utilizzo di queste istruzioni non è sempre appropriato; se il programma si avvale soltanto di alcuni valori diversi è meglio utilizzare le istruzioni CONST, che aumentano la leggibilità. Inoltre, nel caso di dati di ampie dimensioni, l'utilizzo delle istruzioni DATA rende il codice difficile da controllare; il programma dovrebbe leggere i dati da un file, sebbene sia necessario assicurarsi che quest'ultimo sia sempre presente insieme al programma creato.

Errori

Se il valore DATA in fase di lettura in una variabile numerica ha caratteri non numerici, QBasic visualizza il messaggio: "Errore di sintassi". Se un'istruzione READ cerca di leggere dopo l'ultimo valore DATA, e l'istruzione RESTORE non ha ripristinato il puntatore di lettura, QBasic visualizza il messaggio di errore: "Spazio dati esaurito". Se i dati disponibili sono superiori rispetto a quelli effettivamente letti, non viene visualizzato alcun messaggio, ma QBasic si limita a ignorarli.

Esempio

Il programma di seguito riportato legge diversi valori dalle istruzioni DATA nella variabile stringa voce$. Il valore numerico 3.14 viene memorizzato nella variabile voce$ sotto forma di stringa contenente quei caratteri. Gli apici che racchiudono la stringa "contiene, un" consentono di mantenere la virgola; se vengono eliminati, le stringhe "contiene" e "un" vengono considerate come valori di dati separati. Si noti che il valore nullo, specificato da una sola virgola, viene stampato come riga vuota (dato che è una stringa vuota).

Sebbene MAXVALORE sia il nome di una costante, l'istruzione READ la considera come stringa MAXVALORE, non come numero 100. Se venisse letta come valore numerico, potrebbe generare un errore di sintassi, dato che MAXVALORE non può essere considerata come un numero.

```
CLS
CONST MAXVALORE = 100
FOR i% = 1 TO 5
    READ articolo$
    PRINT articolo$
NEXT i%

DATA Sabato,3.14,"contiene, un",,MAXVALORE
```

DIM

Si consulti anche

$DYNAMIC, ERASE, LBOUND, OPTION BASE, REDIM, SHARED, $STATIC, UBOUND

Scopo

L'istruzione DIM consente di dichiarare gli array e di allocare la memoria a essi necessaria; può anche essere utilizzata per specificare il tipo di una variabile.

Sintassi

```
DIM[SHARED] var [(indici)] [AS tipo] [, var[(indici)] [AS tipo]...]
```

La parola chiave facoltativa SHARED indica che alla voce in fase di dichiarazione potranno accedere i sottoprogrammi o le funzioni presenti nel programma.

var è il nome della variabile o dell'array.

indici indica il numero massimo di indici nell'array. Se quest'ultimo ha più di una dimensione, gli indici relativi alle dimensioni devono essere separati da virgole.

AS seguita dal nome di un tipo indica che var consiste di elementi di quel tipo di dati, oppure che è una variabile di quel tipo di dati.

Utilizzo

```
maxX% = 640: maxY% = 480
DIM schermo(maxX%, maxY%)
```

Dichiara un array a due dimensioni con variabili per indici. Dato che gli indici sono variabili, l'array è dinamico.

```
DIM nomi(100) AS STRING * 30
```

Dichiara un array di stringhe a lunghezza fissa.

```
DIM contatore AS LONG
```

Dichiara una variabile di tipo intero lungo.

Descrizione

L'istruzione DIM viene generalmente utilizzata per dichiarare un array, che è una variabile contenente una serie di valori dello stesso tipo. Se viene utilizzata per dichiarare più di un array, è necessario separare le dichiarazioni con virgole.

Il tipo di valori contenuto in un array può essere definito in diversi modi; uno di questi consiste nell'inserire un carattere di dichiarazione del tipo (%, &, !, # o $) alla fine del nome dell'array nell'istruzione DIM. Oppure, è possibile utilizzare la clausola AS, che consente di specificare un tipo definito dall'utente o una stringa a lunghezza fissa. Infine, è anche possibile sfruttare l'istruzione DEFtipo, la quale indica che le variabili (comprese le variabili array) che iniziano con determinate lettere sono di un tipo predefinito.

Il numero o i numeri presenti nelle parentesi subito dopo il nome dell'array indicano il numero di variabili individuali (denominate elementi) dell'array e il numero (denominato indice) che può essere utilizzato per accedere agli elementi. Se viene inserito soltanto un numero, QBasic parte dal presupposto che esso sia il limite superiore degli indici; ad esempio, l'istruzione DIM di seguito riportata dichiara l'array giocatori$ con indici nell'intervallo da 0 a 10.

```
DIM giocatori$(10)
```

Se viene inserita un'istruzione OPTION BASE per modificare da 0 a 1 il limite inferiore di default (la base), gli indici delle istruzioni precedenti variano da 1 a 10.

È possibile specificare sia il limite superiore che quello inferiore degli indici utilizzando la clausola TO tra parentesi, seguita dal nome dell'array. Ad esempio, la seguente istruzione dichiara l'array graduatoria% con gli indici che variano da -10 a 10.

```
DIM graduatoria%(-10 TO 10)
```

Talvolta risulta utile avere più di una dimensione in un array, in questo caso è sufficiente inserire gli indici di ogni dimensione tra parentesi, seguiti dal nome dell'array e separati con virgole. Ad esempio, l'istruzione:

```
DIM foglio(10, 20)
```

dichiara un array che contiene 10 righe e 20 colonne. Sebbene QBasic consenta un numero massimo di 60 dimensioni in un array, è bene non utilizzarne più di tre, dato che un array che disponga di un numero superiore di dimensioni è difficile da concettualizzare.

Se viene inserita la parola chiave SHARED in un'istruzione DIM nel programma principale, è possibile utilizzare l'array specificato in qualsiasi sottoprogramma o

funzione; in caso contrario, un array dichiarato nel programma principale deve essere passato come parametro oppure specificato in un'istruzione SHARED all'interno del sottoprogramma o della funzione.

Gli array dichiarati con letterali o indici costanti sono statici. La memoria per questi array, allocata al momento della compilazione, non può essere modificata durante l'esecuzione.

D'altro canto, gli array dichiarati con gli indici variabili sono dinamici. QBasic alloca la memoria per un array dinamico durante l'esecuzione, consentendo in questo modo di modificare la quantità di memoria utilizzata dall'array, mediante l'istruzione REDIM.

L'istruzione DIM può anche essere utilizzata per dichiarare il tipo di una variabile senza assegnare loro un valore, rendendo il programma più leggibile e consentendo di verificare con più facilità il tipo delle variabili; in questo caso non è possibile inserire i caratteri di dichiarazione del tipo (%, !, #, & e $).

L'istruzione DIM inizializza tutti gli elementi degli array numerici a 0 e tutti gli elementi degli array di stringhe a stringhe nulle. I campi dei tipi definiti dall'utente (comprese le stringhe fisse) vengono inizializzati a 0.

Commenti

Un array di stringhe a lunghezza fissa non può essere passato come parametro a un sottoprogramma o a una funzione; questo limite può essere però superato utilizzando l'istruzione SHARED per permettere a un sottoprogramma di accedere all'array. È anche possibile inserire le stringhe a lunghezza fissa come campi all'interno del tipo di dati definito dall'utente e quindi utilizzare un array di quel determinato tipo.

Errori

Se un programma contiene più di un'istruzione DIM riferita a un array particolare (senza un'istruzione ERASE, nel caso di un array dinamico), oppure se il programma esegue la stessa istruzione DIM più di una volta, QBasic visualizza il messaggio di errore: "Duplice definizione".

Se il valore di un indice non si trova nell'intervallo da -32768 a 32767, QBasic visualizza il messaggio di errore: "Overflow".

Se un array dimensionato implicitamente, oppure un array di stringhe a lunghezza fissa, cerca di utilizzare più memoria di quella disponibile nel segmento dati di default (DGROUP), QBasic visualizza il messaggio di errore: "Spazio dati esaurito" o "Spazio stringhe esaurito". Il primo errore si verifica anche quando il far heap non contiene memoria sufficiente per qualsiasi altro tipo di array. Se le dimensioni di un array memorizzato in DGROUP sono superiori a 32 KB, QBasic visualizza il messaggio di errore: "Matrice troppo grande".

Se viene utilizzato un indice non presente nell'intervallo specificato nelle istruzioni DIM o REDIM, QBasic visualizza il messaggio di errore: "Indice fuori limite", che generalmente indica un errore in un'istruzione di ciclo.

Esempio

Il programma di seguito riportato utilizza l'istruzione DIM per dichiarare due array di tipo intero, uno che contiene i valori di immissione e l'altro il numero di volte in

cui ogni valore viene immesso. Al termine, viene calcolata e visualizzata la *moda* (il valore più frequente).

```
CONST VERO = -1, FALSO = 0

CLS
INPUT "Valore minimo : "; minValore% 'ottiene l'intervallo dei valori di input
INPUT "Valore massimo: "; maxValore%
INPUT "Quanti sono in totale i valori: "; maxInput      'ottiene il numero
                                                        'dei valori inseriti

DIM valori%(maxInput)                       'array per gestire i valori inseriti
DIM frequenza%(minValore% TO maxValore%)    'array per gestire la frequenza di
                                            'ciascun valore

FOR v% = 1 TO maxInput
    DO                                   'ottiene e verifica ciascun valore inserito
        valido = VERO
        PRINT "Inserire valori di tipo intero #"; v%; " ("; minValore%; "-";
         maxValore%; "):";
        INPUT insValore%
        IF INT(insValore%) <> insValore% THEN
            PRINT "Deve essere un intero"
            valido = FALSO
        END IF
        IF insValore% < minValore% OR insValore% > maxValore% THEN
            PRINT "Deve essere compreso tra "; minValore%; " e "; maxValore%;
            ", compresi"
            valido = FALSO
        END IF
    LOOP UNTIL valido = VERO
    valori%(v%) = insValore%
    frequenza%(insValore%) = frequenza%(insValore%) + 1       'annota l'input
NEXT v%

'trova la moda
occorrenze% = 1
FOR v% = minValore% TO maxValore%
    IF frequenza%(v%) > occorrenze% THEN
        occorrenze% = frequenza%(v%)
        valore% = v%
    END IF
NEXT v%

PRINT : PRINT "Valore della moda: ";
FOR v% = minValore% TO maxValore%             'tiene conto di più di una moda
    IF frequenza%(v%) = occorrenze% THEN PRINT v%; " ";
NEXT v%

PRINT : PRINT "occorre "; occorrenze%; " volte"
```

$DYNAMIC

Si consulti anche

$STATIC, REDIM, ERASE

Scopo

Il metacomando $DYNAMIC indica a QBasic di allocare la memoria per la memorizzazione degli array numerici e degli array di stringhe a lunghezza fissa, durante

l'esecuzione del programma. Questo metacomando consente al programma di ridimensionare oppure cancellare gli array durante l'esecuzione, ottenendo un utilizzo più efficace della memoria disponibile.

Sintassi

```
'$DYNAMIC
```

oppure

```
REM $DYNAMIC
```

Utilizzo

```
REM $DYNAMIC
```

Indica al compilatore di trasformare tutti gli array successivi in dinamici.

Descrizione

Se il programma comprende il metacomando $DYNAMIC, QBasic non alloca la memoria negli array fino a quando non inizia l'esecuzione. Utilizzando $DYNAMIC, è possibile assegnare nuovamente lo spazio di memorizzazione degli array, quando il programma richiede una modifica. Le dimensioni di un array possono essere modificate mediante l'istruzione REDIM, oppure è possibile cancellare l'array dalla memoria utilizzando l'istruzione ERASE.

L'opposto di $DYNAMIC è il metacomando $STATIC, che forza QBasic ad allocare lo spazio per gli array durante la fase di compilazione del programma; tuttavia, la memoria allocata negli array statici non può essere rilasciata o ridimensionata. Il metacomando $DYNAMIC deve essere utilizzato quando non si conoscono in anticipo le dimensioni degli array; se invece si sa con esattezza di quanto spazio in memoria necessita un array, l'utilizzo di $STATIC consente di eseguire più rapidamente il programma.

Errori

Se il metacomando $DYNAMIC viene inserito senza la parola chiave REM o un singolo apice, QBasic visualizza il messaggio di errore: "Era atteso: istruzione".

ERASE

Si consulti anche

DIM, $DYNAMIC, REDIM, $STATIC

Scopo

L'istruzione ERASE reinizializza gli elementi di un array statico, oppure rilascia la memoria utilizzata da un array dinamico, consentendo quindi di utilizzare nuovamente un array (oppure la memoria che esso sfrutta), quando il relativo contenuto non è più utile.

Sintassi

```
ERASE array [, array...]
```

array è il nome dell'array da cancellare o da reinizializzare.

Utilizzo

```
imposta% = 4
valPerImposta% = 100
DIM DatiImmessi(imposta%, ValPerImposta%)
.
.
.
ERASE DatiImmessi
```

Dimensiona l'array dinamico DatiImmessi e rilascia la memoria utilizzata dallo stesso.

Descrizione

Nel caso di un array statico, l'istruzione ERASE si limita a ripristinare tutti gli elementi al relativo valore di default, ovvero 0 se numerici, stringhe nulle (""), se stringhe; non modifica le dimensioni di un array statico definito nell'istruzione DIM e non rilascia la memoria.

Nel caso di un array dinamico, ERASE disalloca la memoria utilizzata dall'array, in modo che possa essere sfruttata da altri array dinamici. Prima di poter utilizzare nuovamente un array cancellato, è necessario dimensionarlo nuovamente con DIM. QBasic offre anche l'istruzione REDIM come alternativa rispetto alla cancellazione e al ridimensionamento di un array dinamico: con essa è possibile cancellare, ridimensionare e reinizializzare un array dinamico in una sola fase.

È anche possibile cancellare un array di elementi del tipo di dati definito dall'utente; tutti i campi in ogni elemento, comprese le stringhe a lunghezza fissa, vengono reinizializzati.

Errori

Dopo aver cancellato un array dinamico, qualsiasi riferimento effettuato a un elemento causa il messaggio di errore: "Indice fuori limite". In questo caso è necessario utilizzare innanzitutto l'istruzione DIM o quella REDIM per dimensionare nuovamente l'array. Se viene indicato il nome di un array che non esiste, QBasic visualizza il messaggio di errore: "Matrice non definita".

Esempio

Il programma di seguito riportato dichiara un array dinamico, lo cancella e visualizza la quantità di memoria disponibile per gli array, prima e dopo la cancellazione. Si noti che l'operazione di cancellazione dell'array consente di liberare tutta la memoria da esso utilizzata.

```
DECLARE SUB MostraMemoria ()
CLS

PRINT "Prima di dichiarare l'array: ",
MostraMemoria
```

```
totaleNumeri% = 2500
DIM arrayDinamico$(totaleNumeri%)        'un array dinamico di 2500 stringhe

PRINT "Dopo la dichiarazione dell'array: ",
MostraMemoria

ERASE arrayDinamico$                     'dealloca l'array dinamico
PRINT "Dopo la rimozione dell'array: ",
MostraMemoria
END

SUB MostraMemoria
 'visualizza l'ammontare di memoria disponibile
 PRINT "Lo spazio totale per gli array è di "; FRE(-1)
END SUB
```

LBOUND

Si consulti anche

UBOUND

Scopo

La funzione LBOUND restituisce il limite inferiore (il più piccolo indice disponibile) della dimensione indicata di un array; può essere utilizzata con UBOUND per determinare le dimensioni di un array.

Sintassi

```
LBOUND (array [,dim])
```

array è il nome dell'array.

dim è il numero della dimensione dell'array di cui si deve ottenere il limite inferiore; se non viene indicato, LBOUND restituisce il limite inferiore della prima dimensione.

Utilizzo

```
DIM articoli(50, 100)
inferiore1% = LBOUND(articoli, 1)
inferiore2% = ùLBOUND(articoli, 2)
```

Assegna il valore 0 a inferiore1% e a inferiore2%.

```
DIM freq(500 TO 1000)
inferiore% = LBOUND(freq)
```

Assegna il valore 500 a inferiore%.

Descrizione

La funzione LBOUND restituisce il più piccolo indice disponibile della dimensione indicata di un array; se questa non viene specificata, LBOUND restituisce il valore del limite inferiore della prima dimensione. Per default l'indice minimo di un array è 0, tuttavia l'istruzione OPTION BASE 1 rende 1 il valore minimo, mentre la clausola TO può specificare il limite inferiore, quando si dimensiona o si ridimensiona un array.

Le dimensioni di un array possono essere calcolate sottraendo il valore restituito da LBOUND da quello restituito da UBOUND.

Errori

Se la variabile specificata nell'istruzione LBOUND non è un array, oppure è un array che non è stato dimensionato, QBasic visualizza il messaggio di errore: "Matrice non definita". Se viene specificata una dimensione inferiore a 0 o superiore al numero totale di dimensioni nell'array, oppure se viene indicato un array dinamico che è stato cancellato ma non ridimensionato, QBasic visualizza il messaggio di errore: "Indice fuori limite". Specificando una dimensione 0 non viene generato alcun errore, ma LBOUND restituisce un valore casuale.

Esempio

Il programma di seguito riportato passa un array al sottoprogramma Totale, che utilizza le funzioni LBOUND e UBOUND in un ciclo FOR che accede agli elementi dell'array ed esegue il totale dei relativi valori. Questa operazione è più breve rispetto a quelle di fornire l'indice inferiore o maggiore con il nome dell'array nella chiamata al sottoprogramma.

```
DECLARE SUB Totale (unArray%())
DIM valori%(5 TO 10)

FOR numero% = 5 TO 10
    valori%(numero%) = numero% 'assegna i valori all'array
NEXT numero%

Totale valori%()              'ottiene la somma degli elementi dell'array


SUB Totale (unArray%())
    t = 0
    FOR n% = LBOUND(unArray%) TO UBOUND(unArray%)
        t = t + unArray%(n%)
    NEXT n%
    PRINT "Totale dei valori dell'array: "; t
END SUB
```

OPTION BASE

Si consulti anche

DIM, LBOUND, UBOUND

Scopo

L'istruzione OPTION BASE indica il limite inferiore di default per gli indici di un array; questo valore può essere 0 o 1.

Sintassi

```
OPTION BASE [0 | 1]
```

Utilizzo

```
OPTION BASE 1
```

Imposta l'indice inferiore di default a 1.

Descrizione

L'istruzione OPTION BASE deve essere utilizzata con il valore 0 o 1 per impostare il limite inferiore di default di un array. Per default, quando viene utilizzato un array non dichiarato in un'istruzione DIM o non dimensionato con la clausola TO, il limite inferiore di default per ogni dimensione è 0. Utilizzando l'istruzione:

```
OPTION BASE 1
```

1 diventa il primo indice di ogni dimensione. Sebbene in molti linguaggi gli indici degli array inizino da 0, spesso risulta più naturale etichettare il primo elemento di un elenco come numero 1 e quindi utilizzare l'indice 1 come il limite inferiore di un array che contiene l'elenco. Quindi, se viene progettato un codice che non utilizza mai l'indice 0, l'istruzione OPTION BASE 1 offre due vantaggi: il primo consiste nel fatto che impedisce di utilizzare inavvertitamente l'indice 0 (generando il messaggio di errore: "Indice fuori limite"); il secondo è che consente di risparmiare memoria non allocando lo spazio per l'elemento non utilizzato, ovvero il valore 0.

Quando viene inserita la clausola TO per ogni dimensione in un'istruzione DIM, è possibile specificare il limite inferiore e quello superiore di un array. Qualsiasi limite indicato in una clausola TO elimina il limite inferiore di default; dato che quest'ultimo è 0, l'istruzione OPTION BASE 0 risulta utile soltanto per indicare a coloro che utilizzano il programma che l'indice inferiore è 0.

Errori

Se viene utilizzato un valore diverso da 0 o da 1 con l'istruzione OPTION BASE, QBasic visualizza il messaggio di errore: "Era atteso: 0 o 1". Se si cerca di utilizzare un'altra istruzione OPTION BASE nello stesso programma, QBasic visualizza il messaggio di errore: "Matrice già dimensionata".

Avvertenze

QBasic ignora l'istruzione OPTION BASE in un programma concatenato, se un array dinamico viene passato al programma mediante un'istruzione COMMON. In questo caso, il programma concatenato eredita il limite inferiore di default dal programma a cui si concatena.

Esempio

Il programma di seguito riportato illustra in che modo l'istruzione OPTION BASE 1 assegni il valore 1 all'indice inferiore dell'array.

```
OPTION BASE 1           'imposta a 1 l'indice del primo elemento

DIM matrice(10, 10)
PRINT "Il primo elemento nell'array bidimensionale è matrice"
PRINT "("; LBOUND(matrice, 1); ","; LBOUND(matrice, 2); ")"
```

READ

Si consulti anche

DATA, RESTORE

Scopo

L'istruzione READ consente di accedere ai valori di una o più istruzioni DATA e di assegnare tali valori alle variabili indicate; viene generalmente utilizzata all'interno dei cicli per leggere i valori in un array.

Sintassi

```
READ var [, var...]
```

var è una variabile numerica o stringa di qualsiasi tipo. Se sono presenti diverse variabili, è necessario separarle con virgole (gli array e i tipi definiti dall'utente devono essere letti un elemento o un campo alla volta).

Utilizzo

```
READ n$, a$, num
DATA Franchi, Milano, 1
```

Legge i valori "Franchi", "Milano" e 1 nelle variabili n$, a$ e num.

Descrizione

L'istruzione READ legge i valori in un'istruzione DATA individuata nel programma principale e li memorizza nelle variabili indicate; i valori vengono letti nell'ordine in cui appaiono nelle istruzioni DATA (si noti che è possibile utilizzare l'istruzione RESTORE per leggere nuovamente i valori partendo dall'inizio dell'istruzione DATA).

Quando i valori vengono assegnati alle variabili, l'istruzione READ li converte nel tipo di dati della variabile, nel modo seguente.

- I valori numerici non interi letti nelle variabili di tipo intero vengono arrotondati all'intero più vicino.
- Le variabili numeriche lette in variabili stringa vengono memorizzate sotto forma di stringhe di cifre.
- I valori stringa letti nelle variabili stringa a lunghezza fissa vengono troncati, se sono troppo lunghi; se invece sono troppo corti, vengono giustificati a sinistra e riempiti a destra con gli spazi necessari.

Errori

Se si cerca di leggere più valori di quelli disponibili nelle istruzioni DATA, QBasic visualizza il messaggio di errore: "Spazio dati esaurito". Se viene letto un numero inferiore di dati rispetto a quelli disponibili, la successiva istruzione READ (se presente) inizia la lettura in corrispondenza del primo valore non ancora letto. QBasic non controlla che tutti i valori indicati nelle istruzioni DATA vengano letti. Se si cerca di leggere un valore di stringa in una variabile numerica, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Avvertenze

Non è possibile leggere un intero record (tipo di dati definito dall'utente) in una sola variabile: Se RecCliente è una variabile del tipo definito dall'utente, non è possibile

utilizzare l'istruzione READ RecCliente; in questo caso è necessario leggere i campi uno alla volta, ad esempio:

```
READ RecCliente.nome RecCliente.via, ReCcliente.citta
```

Esempio

Il programma di seguito riportato utilizza un'istruzione READ per leggere gli interi in un array. I valori vengono quindi impiegati per tracciare una stella.

```
DIM punti%(28)
FOR pt% = 1 TO 28
    READ punti%(pt%)          'legge i valori dei dati e li assegna a un array
NEXT pt%

'disegna la stella
SCREEN 1
PSET (punti%(1), punti%(2))
FOR pt% = 3 TO 28 STEP 2
    LINE -(punti%(pt%), punti%(pt% + 1))
NEXT pt%

DATA 100,100,150,100,175,75,200,100
DATA 250,100,200,125,250,150,200,150
DATA 175,175,150,150,125,150,100,150
DATA 150,125,100,100
```

REDIM

Si consulti anche

DIM, ERASE

Scopo

L'istruzione REDIM modifica lo spazio assegnato, ridimensiona e reinizializza un array dinamico, consentendo in questo modo di utilizzare durante l'elaborazione lo stesso array per gruppi diversi di dati.

Sintassi

```
REDIM [SHARED] var(indici) [AS tipo] [, var(indici) [AS tipo]...]
```

La parola chiave facoltativa SHARED indica che i sottoprogrammi e le funzioni presenti nel programma possono accedere agli array dichiarati.

var è il nome della variabile array dinamica da ridimensionare.

indici indica gli indici per ogni dimensione dell'array. Se quest'ultimo ha più di una dimensione, gli indici devono essere separati da virgole. La clausola AS può indicare il tipo di dati contenuti nell'array.

Utilizzo

```
righe% = 10: col% = 10
DIM matrice(righe%, col%)
.
```

```
.
.
righe = 20: col = 15
REDIM matrice(righe, col)
```

Ridimensiona l'array matrice.

Descrizione

L'istruzione REDIM può ridimensionare soltanto gli array dinamici (ovvero quelli che utilizzano variabili come indici nell'istruzione DIM o che sono resi dinamici con il metacomando $DYNAMIC). QBasic indica le dimensioni di questi array e alloca la memoria necessaria al momento dell'esecuzione, invece che durante la compilazione, come avviene con gli array statici. Dopo aver utilizzato DIM o $DYNAMIC per definire un array dinamico, è possibile utilizzare REDIM in un punto qualsiasi del programma per modificare lo spazio assegnato e gli intervalli degli indici.

La sintassi di REDIM è analoga a quella di DIM: per rendere accessibile l'array a tutti i sottoprogrammi e le funzioni è necessario inserire dopo l'istruzione REDIM la parola chiave SHARED, altrimenti si deve passare l'array come parametro a un sottoprogramma o utilizzare l'istruzione SHARED all'interno di un sottoprogramma o di una funzione.

Come per DIM, è possibile indicare più di un array nella stessa istruzione REDIM, separandoli con virgole.

Errori

Se si cerca di modificare il numero delle dimensioni in un array ridimensionato, QBasic visualizza il messaggio di errore: "Numero di dimensioni errato". Se il tipo di dati memorizzato nell'array viene modificato, QBasic visualizza il messaggio di errore: "Duplice definizione".

Avvertenze

L'istruzione REDIM non può essere utilizzata con un array statico. Inoltre, non è possibile modificare il numero delle dimensioni dell'array o il tipo di dati (intero, doppio, tipo definito dall'utente e così via).

Esempio

Il programma di seguito riportato dichiara e riempie un array e quindi utilizza l'istruzione REDIM per ridimensionarlo, visualizzando il contenuto dell'array prima e dopo l'esecuzione di questa operazione. Il programma illustra inoltre il passaggio di un array a un sottoprogramma e l'utilizzo LBOUND e UBOUND per accedere correttamente a ogni elemento.

```
DECLARE SUB RiempiMatrice (unArray%())
DECLARE SUB StampaMatrice (unArray%())

CLS
OPTION BASE 1            'imposta a 1 l'indice del primo elemento dell'array
righe% = 5: colonne% = 5
DIM matrice%(righe%, colonne%)         'dichiara un array dinamico
RiempiMatrice matrice%()               'riempie l'array
StampaMatrice matrice%()               'visualizza l'array
```

```
righe% = 6: colonne% = 8
REDIM matrice%(righe%, colonne%)          'ridimensiona l'array
RiempiMatrice matrice%()                  'riempie l'array
PRINT : PRINT
StampaMatrice matrice%()                  'visualizza l'array


SUB RiempiMatrice (unArray%())
    'riempie un array bi-dimensionale
    FOR righe% = LBOUND(unArray%, 1) TO UBOUND(unArray%, 1)
        FOR colonne% = LBOUND(unArray%, 2) TO UBOUND(unArray%, 2)
            unArray%(righe%, colonne%) = colonne%
        NEXT colonne%
    NEXT righe%
END SUB

SUB StampaMatrice (unArray%())
    'visualizza i contenuti di qualunque array bidimensionale
    FOR righe% = LBOUND(unArray%, 1) TO UBOUND(unArray%, 1)
        FOR colonne% = LBOUND(unArray%, 2) TO UBOUND(unArray%, 2)
            PRINT unArray%(righe%, colonne%); " ";
        NEXT colonne%
    PRINT
    NEXT righe%
END SUB
```

RESTORE

Si consulti anche

DATA, READ

Scopo

L'istruzione RESTORE consente di utilizzare nuovamente con READ tutti i dati letti in precedenza dalle istruzioni DATA, oppure di rileggere le istruzioni DATA partendo da una determinata posizione. In questo modo è possibile leggere nuovamente i dati partendo dall'inizio delle istruzioni DATA oppure leggerli in modo selezionato, in base a una determinata condizione.

Sintassi

```
RESTORE [riga | etichetta]
```

riga o etichetta indicano la riga che contiene l'istruzione DATA da leggere successivamente.

Utilizzo

```
RESTORE
```

Legge nuovamente i dati partendo dall'inizio della prima istruzione DATA.

```
RESTORE Lunedi
```

Legge nuovamente i dati dall'istruzione DATA sulla riga etichettata "Lunedi".

Descrizione

Generalmente, l'istruzione READ consente di leggere i dati partendo dalla prima voce della prima istruzione DATA; le successive istruzioni READ operano in modo sequenziale. L'istruzione RESTORE consente di definire un nuovo punto di inizio per la lettura dei dati. Se questa istruzione viene utilizzata senza un numero di riga o un'etichetta di riga, la successiva istruzione READ può ottenere l'accesso ai dati presenti all'inizio della prima istruzione DATA. Questo risulta utile per reinizializzare i dati, prima di avviare nuovamente l'elaborazione.

Se RESTORE viene utilizzata con un numero di riga o un'etichetta di riga, la successiva istruzione READ legge i dati presenti all'inizio dell'istruzione DATA sulla riga con il numero o l'etichetta indicati. Si noti che RESTORE non legge nulla, ma si limita a specificare l'istruzione DATA che deve essere letta dalla successiva istruzione READ.

Errori

Se in un'istruzione RESTORE viene specificata un'etichetta o un numero di riga non esistente, QBasic visualizza il messaggio di errore: "Etichetta non definita".

Esempio

Il programma di seguito riportato etichetta tre diverse istruzioni DATA e legge il valore di salario! da una di queste, in base alla scelta immessa dall'utente. In pratica si potrebbero definire delle costanti con l'istruzione CONST, per le voci nelle istruzioni DATA di questo esempio. Tuttavia, se si dispone di ampi gruppi di voci di dati è meglio utilizzare istruzioni DATA etichettate perché richiedono meno memoria.

```
CLS
PRINT "Salario Regolare, Straordinario, o Doppio: "
INPUT "R, S, o D"; tipo$
SELECT CASE UCASE$(tipo$)
    CASE "R": RESTORE Regolare
    CASE "S": RESTORE Straordinario
    CASE "D": RESTORE Doppio
END SELECT

READ salario!
PRINT "Il tuo salario orario è ";
PRINT USING "L. #####"; salario!

Regolare:
DATA 10000

Straordinario:
DATA 15000

Doppio:
DATA 20000
```

$STATIC

Si consulti anche

$DYNAMIC, REDIM, ERASE

Scopo

Il metacomando $STATIC forza a QBasic ad allocare la memoria per la memorizzazione degli array durante la compilazione del programma. Gli array statici permettono un'esecuzione più rapida del programma, ma, dato che non possono essere ridimensionati o cancellati, possono causare un utilizzo non efficace della memoria.

Sintassi

```
'$STATIC
```

oppure

```
REM $STATIC
```

Utilizzo

```
REM $STATIC
```

Indica al compilatore di trasformare tutti gli array successivi in statici.

Descrizione

Se il programma comprende il metacomando $STATIC, QBasic alloca lo spazio di memorizzazione per gli array dimensionati esplicitamente (ovvero quelli per cui viene specificato un determinato numero di elementi nell'istruzione DIM), durante la compilazione del programma.

Gli array dimensionati da variabili o espressioni i cui valori non sono conosciuti in anticipo sono array dinamici, per i quali QBasic alloca la memoria necessaria dopo che il programma ha iniziato l'esecuzione. Dato che QBasic alloca la memoria per gli array statici prima dell'inizio di un programma, questo non deve creare gli array durante l'esecuzione; ne consegue che un programma che utilizza array statici viene eseguito più velocemente rispetto a un programma che si avvale di array dinamici.

La memoria assegnata agli array statici non può essere rilasciata o ridimensionata durante l'esecuzione del programma. L'istruzione REDIM non può essere utilizzata per modificare il numero degli elementi in un array statico, dato che la quantità di memoria allocata non può essere modificata. L'istruzione ERASE non libera la memoria utilizzata da un array statico, anche se ripristina a 0 o a stringhe nulle tutti gli elementi di un array.

Se il programma fa riferimento a un array che non è stato esplicitamente dimensionato da una precedente istruzione DIM o REDIM, QBasic crea l'array con 11 elementi (numerati da 0 a 10). Questi array, denominati array dimensionati implicitamente, sono sempre statici e quindi non possono essere ridimensionati o cancellati.

É indispensabile non confondere il metacomando $STATIC con l'istruzione STATIC.

Errori

Se il metacomando $STATIC viene inserito senza la parola chiave REM o un singolo apice, QBasic visualizza il messaggio di errore: "Era atteso: espressione".

UBOUND

Si consulti anche

LBOUND

Scopo

La funzione UBOUND restituisce il limite superiore (l'indice più grande disponibile) della dimensione indicata di un array; può essere utilizzata con LBOUND per determinare le dimensioni di un array.

Sintassi

```
UBOUND (array [,dim])
```

array è il nome dell'array.

dim è il numero della dimensione dell'array di cui si deve ottenere il limite superiore; se non viene indicato, UBOUND restituisce il limite superiore della prima dimensione.

Utilizzo

```
DIM articoli(50, 100)
superiore1% = UBOUND(articoli, 1)
superiore2% = UBOUND(articoli, 2)
```

Assegna il valore 50 a superiore1% e il valore 100 a superiore2%.

```
DIM freq(500 TO 1000)
superiore% = UBOUND(freq)
```

Assegna il valore 1000 a superiore%.

Descrizione

La funzione UBOUND restituisce l'indice più grande disponibile della dimensione indicata di un array. Se questa non viene specificata, UBOUND restituisce il valore del limite superiore della prima dimensione.

Le dimensioni di un array possono essere calcolate sottraendo il valore restituito a LBOUND da quello restituito da UBOUND. Questa funzione risulta utile per l'individuazione del limite superiore di un ciclo che deve gestire ogni elemento di un array.

Errori

Se la variabile specificata nell'istruzione UBOUND non è un array, oppure è un array che non è stato dimensionato, QBasic visualizza il messaggio di errore: "Matrice non definita". Se viene specificata una dimensione inferiore a 0 o superiore al numero totale di dimensioni nell'array, oppure se viene indicato un array dinamico che è stato cancellato ma non ridimensionato, QBasic visualizza il messaggio di errore: "Indice fuori limite". Specificando una dimensione 0 non viene generato alcun errore, ma UBOUND restituisce un valore casuale.

Esempio

Il programma di seguito riportato utilizza la funzione definita dall'utente DimensioneDi per determinare il numero totale di byte necessari alla memorizzazione di un array di tipo intero. Quando il programma richiama la funzione, passa a quest'ultima il nome dell'array e il numero delle dimensioni.

Per quanto riguarda l'array di questo programma, viene visualizzato il messaggio: "L'array utilizza 1890 byte". Si noti che, dato che il limite inferiore non viene specificato e non viene utilizzata alcuna istruzione OPTION BASE, il limite inferiore di ogni dimensione è 0. Quindi, il numero totale di elementi di tipo intero è dato da 9 per 7 per 5 per 3; la funzione moltiplica questo valore per 2, in modo da determinare il numero totale di byte occupati dall'array.

```
DECLARE FUNCTION DimensioneDi (unArray%(), dimensioni%)

DIM IntArray%(8, 6, 4, 2)
PRINT "L'array utilizza"; DimensioneDi(IntArray%(), 4); "byte."

FUNCTION DimensioneDi (unArray%(), dimensioni%)
    'restituisce le dimensioni di un array di interi
    dimensione = 1                   'gestisce il prodotto delle dimensioni
    FOR dimen% = 1 TO dimensioni%
        subDimensione% = UBOUND(unArray%, dimen%) - LBOUND(unArray%, dimen%) + 1
        dimensione = dimensione * subDimensione%
    NEXT dimen%
    DimensioneDi = dimensione * 2        '2 byte per intero
END FUNCTION
```

Capitolo 7

Funzioni matematiche

INTRODUZIONE

La maggior parte delle applicazioni informatiche utilizza soltanto le operazioni matematiche standard, ovvero addizione, sottrazione, moltiplicazione e divisione. In effetti, è possibile scrivere programmi validi senza dover necessariamente conoscere la matematica, a parte l'aritmetica di base.

Nondimeno, le funzioni matematiche più avanzate proprie di QBasic risultano utili, e talvolta indispensabili, in molte applicazioni. Le funzioni trigonometriche sono importanti nei programmi di CAD (Computer Aided Design, ovvero progettazione tramite computer); molti giochi e simulazioni necessitano di una sorgente di numeri casuali. Le applicazioni commerciali richiedono la capacità di arrotondare i numeri a un determinato numero di cifre, mentre le applicazioni scientifiche devono spesso determinare i risultati dei calcoli con molte cifre di precisione. In questo capitolo vengono esaminate le possibilità offerte dalle funzioni matematiche di QBasic e vengono inoltre dati alcuni suggerimenti relativi al loro utilizzo. Nella Tabella 7.1 sono riportate in ordine alfabetico le funzioni e le istruzioni matematiche.

FUNZIONI TRIGONOMETRICHE E RADICE QUADRATA

QBasic comprende quattro funzioni trigonometriche: ATN (arcotangente), COS (coseno), SIN (seno) e TAN (tangente). I valori restituiti da queste funzioni dipendono dalle relazioni tra un angolo e i lati di un triangolo rettangolo, come illustrato nella Figura 7.1.

Le funzioni SIN, COS e TAN gestiscono gli angoli espressi in radianti (un *radiante* è una misura angolare uguale all'angolo formato nel centro di un cerchio da un arco di lunghezza uguale al raggio del cerchio). Per convertire i gradi in radianti, è sufficiente moltiplicare il numero di gradi per il risultato di π diviso per 180. Per

Tabella 7.1 Le funzioni e le istruzioni matematiche di QBasic

Istruzione o funzione	Significato
ABS	Restituisce il valore assoluto (positivo) di un numero
ATN	Restituisce un angolo, in radianti, la cui tangente è il numero fornito
CDBL	Restituisce un numero come valore a precisione doppia
CINT	Restituisce un numero arrotondato all'intero più vicino
CLNG	Restituisce un numero come valore intero lungo
COS	Restituisce il coseno di un angolo misurato in radianti
CSNG	Restituisce un numero come valore a precisione semplice
EXP	Restituisce il numero *e* (la base naturale, che è uguale a 2.718282) elevato alla potenza indicata
FIX	Restituisce la parte intera di un numero
INT	Restituisce il massimo valore intero minore o uguale al numero indicato
LOG	Restituisce il logaritmo naturale (base e) di un numero
RANDOMIZE	Inizializza il generatore di numeri casuali (definisce una sequenza di numeri casuali)
RND	Restituisce un valore da una sequenza di numeri casuali
SGN	Restituisce il segno di un numero (-1, 0 o 1)
SIN	Restituisce il seno di un angolo misurato in radianti
SQR	Restituisce la radice quadrata di un numero
TAN	Restituisce la tangente di un angolo misurato in radianti

convertire i radianti in gradi, è necessario moltiplicare il numero di radianti per il risultato di 180 diviso per π (approssimativamente 57.2958). Le costanti di seguito riportate potrebbero essere utili nei programmi che eseguono la conversione da gradi in radianti.

```
CONST PI = 3.141593
CONST GRADIARAD = PI / 180
```

Se fosse necessaria una maggiore precisione, è possibile definire PI come costante a precisione doppia e aggiungere due cifre al valore indicato. A questo punto, sapendo quanti gradi vi sono in un angolo, è possibile utilizzare l'espressione angolo * GRADIARAD per passare da gradi in radianti, come riportato nel seguente calcolo:

```
risultato = SIN(angolo * GRADIARAD)
```

Questo esempio esprime il valore in precisione semplice. Le funzioni trigonometriche restituiscono per default valori a precisione semplice, anche se, specificando un valore a precisione doppia, anche il risultato fornito è a precisione doppia.

Le funzioni trigonometriche sono utili per calcolare una lunghezza o un'altezza sconosciuta quando si conosce un angolo e la lunghezza di un lato di un triangolo rettangolo. Ad esempio, utilizzando la Figura 7.1 come guida, è possibile individuare l'altezza di un albero senza per questo doverlo scalare. Si pianti un picchetto come

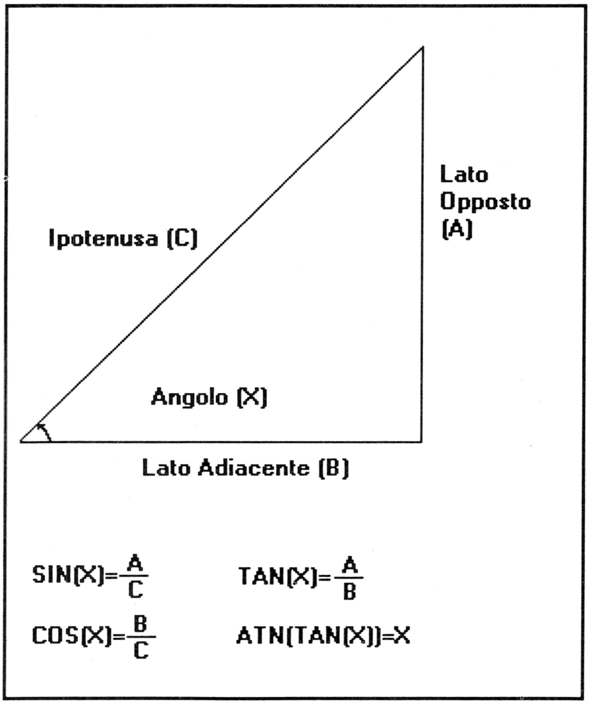

Figura 7.1 Le funzioni trigonometriche

vertice per l'angolo X nel terreno a pochi passi dall'albero e quindi si utilizzi un teodolite (un particolare strumento per la misura degli angoli) per misurare l'angolo formato con la cima dell'albero. Poi, si tracci una riga per misurare il lato B (la distanza dalla base dell'albero). A questo punto QBasic può calcolare l'altezza dell'albero, come dimostrano le istruzioni di seguito riportate.

```
INPUT "Angolo in gradi: ";angoloX        'richiede l'angolo
INPUT "Lunghezza del lato: "lunghB       'richiede il lato conosciuto
radiantiX = angoloX * GRADIARAD          'converte l'angolo in radianti
tanX = TAN(radiantiX)                    'calcola la tangente dell'angolo X
altezza = tanX * lunghB                  'calcola l'altezza
```

Questo esempio utilizza l'algebra e la trigonometria per individuare l'altezza dell'albero. Dato che TAN(X) è uguale ad A diviso per B, A (l'altezza dell'albero) è uguale a TAN(X) * B.

Dalle quattro funzioni trigonometriche fornite da QBasic è possibile ricavarne molte altre; nella Tabella 7.2 sono riportati alcuni esempi, in cui A è un angolo in radianti. La funzione SQR, utilizzata in alcune delle formule presenti in questa tabella, restituisce la radice quadrata di qualsiasi valore maggiore o uguale a 0.

Tabella 7.2 Funzioni trigonometriche derivate

Funzione	Formula
Secante	SEC(A) = 1 / COS(A)
Cosecante	CSC(A) = 1 / SIN(A)
Cotangente	COT(A) = 1 / TAN(A)
Seno inverso	ARCSIN(A) = ATN(A / SQR(-A * A + 1))
Coseno inverso	ARCCOS(A) = PI/2 - ATN(A / SQR(-A * A + 1))
Secante inversa	ARCSEC(A) = ATN(SQR(A * A - 1)) + SGN(SGN(A) - 1) * PI
Cosencante inversa	ARCCSC(A) = ATN(1 / SQR(A * A - 1)) + SGN(SGN(A) - 1) *PI
Cotangente inversa	ARCCOT(A) = ATN(1 / A) - SGN(SGN(A) - 1) * PI
Seno iperbolico	SINH(A) = (EXP(A) - EXP(-A)) / 2
Coseno iperbolico	COSH(A) = (EXP(A) + EXP(-A)) / 2
Tangente iperbolica	TANH(A) = (EXP(A) - EXP(-A)) / (EXP(A) + EXP(-A))
Secante iperbolica	SECH(A) = 2 / (EXP(A) + EXP(A))
Cosecante iperbolica	CSCH(A) = 2 / (EXP(A) - EXP(-A))
Cotangente iperbolica	COTH(A) = (EXP(A) + EXP(-A)) / (EXP(A) - EXP(-A))

LOGARITMI ED ESPONENZIALI

Due funzioni di QBasic eseguono le operazioni con i logaritmi e gli esponenziali: la funzione LOG determina il *logaritmo naturale* di un numero o di un'espressione indicati, ovvero l'esponente a cui deve essere elevata la base e (2.718282) per uguagliare il valore indicato; la funzione EXP rappresenta il complemento di LOG, eleva e al valore specificato. Per convertire i logaritmi naturali in logaritmi comuni, a base 10, è sufficiente dividere il logaritmo naturale del numero per il logaritmo naturale di 10, come nella funzione di seguito riportata.

```
FUNCTION LOG10(numero)
   LOG10 = LOG(numero) / LOG(10)
END FUNCTION
```

I logaritmi vengono generalmente utilizzati nei modelli scientifici e nelle simulazioni, ad esempio quelle relative alla crescita della popolazione.

FUNZIONI DI CONVERSIONE NUMERICA

Alcune funzioni matematiche consentono di convertire i numeri da un tipo a un altro, ad esempio quelli a precisione semplice in precisione doppia, oppure quelli interi in interi lunghi. Questo gruppo di funzioni consente di convertire un valore numerico nella forma prevista da una funzione o da un sottoprogramma; ad esempio, se la funzione Calc prevede due valori a precisione doppia come parametri, ma finora si è lavorato con le variabili a precisione semplice num1! e num2!, è possibile chiamare la funzione soltanto convertendo le variabili, come riportato nelle seguenti istruzioni:

```
risultato# = Calc(CDBL(num1!), CDBL(num2!))
```

Per convertire un numero in precisione doppia, è sufficiente utilizzare la funzione CDBL; per convertirne uno in precisione semplice, occorre utilizzare la funzione CSNG. Per convertire un numero in un intero è necessario sfruttare la funzione CINT, per convertirlo in un intero lungo invece, la funzione CLNG. Dato che la precisione aumenta con il numero di cifre significative, la conversione di un numero a una precisione inferiore potrebbe ridurre l'accuratezza dello stesso. D'altro canto, anche se la conversione di un numero a una precisione superiore aggiunge più posizioni decimali, non necessariamente questo rende il numero più preciso.

TRONCAMENTO E ARROTONDAMENTO

Tre sono le funzioni che permettono di arrotondare o di troncare i numeri. Le funzioni INT e FIX troncano i numeri eliminando le cifre alla destra del punto decimale e restituiscono un intero, ad esempio INT(3.445) è uguale a 3. Queste funzioni si differenziano soltanto nel modo di gestione dei numeri negativi: INT restituisce il massimo intero inferiore, ad esempio INT(-3.52) è uguale a -4; FIX restituisce il minimo intero superiore, ad esempio FIX(-3.25) è uguale a -3. Il trocamento risulta utile quando si desidera soltanto la parte intera di un numero.

La funzione CINT ("conversione in intero") consente di arrotondare un valore all'intero più vicino: un valore decimale positivo maggiore di 0.5, per i numeri positivi, viene arrotondato all'intero successivo, ad esempio CINT(5.52) è uguale a 6, ma CINT(5.49) è uguale a 5; un valore decimale maggiore di 0.5, per i numeri negativi, viene arrotondato al massimo intero inferiore: CINT(-5.52) è uguale a -6, ma CINT(-5.49) è uguale a -5 (una frazione che vale esattamente 0.5 viene arrotondata all'intero pari più vicino). La funzione CINT deve essere utilizzata quando non si desidera gestire le frazioni, ma mantenere un altro grado di precisione.

NUMERI CASUALI

Le ultime due funzioni affrontate in questo capitolo forniscono ai programmatori i mezzi per creare i numeri casuali; un *numero casuale* è semplicemente un numero che non ha alcuna relazione con i propri predecessori. Questi numeri risultano utili per molti giochi, in cui assolvono lo stesso compito dei dadi; anche le simulazioni li utilizzano per modellare un comportamento imprevedibile. La parte di programma di seguito riportata utilizza l'istruzione RANDOMIZE e la funzione RND per stampare cinque numeri casuali.

```
RANDOMIZE 10

FOR casoNum% = 1 TO 5
   PRINT RND
NEXT casoNum%
```

Questo programma visualizza i seguenti valori:

```
.5749933
.2375866
.5295308
.2520258
.9770579
```

L'istruzione RANDOMIZE inizializza il generatore di numeri casuali con un valore utilizzato in una formula che crea risultati imprevisti, comunque ogni valore iniziale specifico genera sempre la stessa sequenza di numeri casuali. Sebbene questo tipo di casualità risulti utile per la verifica di una simulazione o di un gioco, un gioco effettivo richiede che i numeri siano imprevedibili e matematicamente casuali. Per creare una sequenza diversa di numeri casuali ogni volta che viene eseguito un programma, è sufficiente utilizzare l'istruzione RANDOMIZE TIMER. La funzione TIMER restituisce il numero di secondi contati dall'orologio di sistema partendo dalla mezzanotte precedente, fornendo in questo modo all'istruzione RANDOMIZE un valore ragionevolmente imprevedibile.

Bibliografia

Graham, Ronald L., Donald E. Knuth and Oren Patashnik. *Concrete Mathematics.* Reading, Mass: Addison-Wesley, 1989.

Hewitt, Paul G. *Conceptual Physics.* Boston, Mass: Little and Brown, 1985.

ABS

Si consulti anche

SNG

Scopo

La funzione ABS restituisce il valore assoluto di un'espressione numerica, ovvero l'effettiva quantità del numero, indipendentemente dal segno; ad esempio, il valore assoluto di -1 è 1. Il valore assoluto è sempre un numero positivo.

Sintassi

```
ABS(espr)
```

espr è una costante numerica, una variabile numerica oppure un'espressione numerica che può essere di qualsiasi tipo (intero, intero lungo, a precisione semplice, a precisione doppia).

Utilizzo

```
PRINT ABS(-45)
```

Visualizza 45, ovvero il valore assoluto di -45.

```
differenza = ABS(ipotesi1 - ipotesi2)
```

Assegna alla variabile differenza il valore della differenza assoluta tra i valori di ipotesi1 e ipotesi2.

Descrizione

La funzione ABS restituisce il valore senza segno di un'espressione numerica, che può comprendere variabili o costanti di qualsiasi tipo numerico. La precisione del valore restituito da ABS è identica a quella utilizzata nell'espressione; ad esempio, ABS(-1) restituisce 1, mentre ABS(99.44555) restituisce 99.44555.

La funzione ABS viene spesso sfruttata nelle routine che utilizzano l'approssimazione; ad esempio, un programma che calcola le radici cubiche potrebbe confrontare il cubo del valore attuale calcolato con quello originale e accettare il calcolo quando il valore assoluto della differenza tra i due valori è inferiore rispetto alla tolleranza specificata, come si può vedere nell'istruzione DO...LOOP di seguito riportata.

```
DO UNTIL ABS(destinazione - risultato) < tolleranza
   'esegue il calcolo
LOOP
```

Questo tipo di approssimazione ha molte altre applicazioni. Ad esempio, un programma potrebbe leggere le coordinate x e y di una pallina sullo schermo e "farla rimbalzare" quando il valore assoluto della sua distanza dal bordo dello schermo è sufficientemente piccolo.

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato genera un "obiettivo" casuale su una griglia quadrata di 100 per 100 e richiede di ipotizzare le coordinate x e y. Il programma visualizza la differenza tra l'ipotesi e l'obiettivo per indicare quanto ci si è avvicinati; quando la differenza tra questi due valori è inferiore a 5, sia per la coordinata x che per quella y, l'obiettivo è da considerarsi colpito.

```
RANDOMIZE TIMER                   'ottiene sequenze di numeri casuali
DEFINT A-Z                        'definisce tutte le variabili di tipo intero
obiettivoX = RND * 100            'coordinata X casuale tra 0 e 99
obiettivoY = RND * 100            'coordinata Y casuale tra 0 e 99
intorno = 5                       'distanza entro la quale il bersaglio è da
considerarsi colpito
distanzaX = 100                   'distanza di x dall'obiettivo
distanzaY = 100                   'distanza di y dall'obiettivo

CLS
PRINT "Tenta di colpire un obiettivo in una griglia di quadrati 100 per 100":
PRINT
DO UNTIL distanzaX <= intorno AND distanzaY <= intorno
    INPUT "Coordinata di tiro X "; xTiro
    INPUT "Coordinata di tiro Y "; yTiro
    distanzaX = ABS(obiettivoX - xTiro)        'ottiene la distanza assoluta
    distanzaY = ABS(obiettivoY - yTiro)
    PRINT "La distanza x+y dall'obiettivo è "; distanzaX + distanzaY: PRINT
LOOP
```

```
PRINT "Hai colpito l'obiettivo!"        'le coordinate x e y sono nell'intorno
PRINT "L'obiettivo era a x= "; obiettivoX; "y= "; obiettivoY
```

ATN

Si consulti anche

COS, SIN, TAN

Scopo

La funzione ATN restituisce l'arcotangente di un'espressione numerica, ovvero l'angolo (misurato in radianti) la cui tangente è uguale al valore specificato.

Sintassi

```
ATN(espr)
```

espr può essere una qualsiasi combinazione di numeri o variabili numeriche (interi, interi lunghi, a precisione semplice, a precisione doppia). Il risultato viene espresso in radianti, che possono essere convertiti in gradi moltiplicando per 180 diviso per π, approssimativamente 57.2958.

Utilizzo

```
PRINT ATN(.8000001)
```

Visualizza l'arcotangente di .8000001 in precisione doppia, ovvero .6747410031991594 radianti.

Descrizione

La funzione ATN restituisce la misura di un angolo (in radianti), la cui tangente è il valore dell'espressione specificata. La funzione restituisce per default un valore a precisione semplice in virgola mobile, se però il valore fornito è a precisione doppia, anche l'arcotangente è un valore analogo. Il valore restituito da ATN si trova sempre compreso tra -π/2 e π/2 (il valore di π è approssimativamente 3.141593).

La funzione ATN (come le altre funzioni trigonometriche) risulta utile nella soluzione di problemi relativi alla geometria, alla trigonometria o alla topografia. Ad esempio, se si conoscono le lunghezze dei due lati di un triangolo rettangolo è possibile ottenere la tangente di un angolo dividendo la lunghezza del lato opposto per quella del lato adiacente. Come illustrato nell'istruzione di seguito riportata, è possibile determinare la misura di un angolo in radianti calcolando l'arcotangente della tangente, e in seguito è possibile convertire il risultato in gradi.

```
angoloInRadianti = ATN(LatoOpposto / LatoAdiacente)
angoloInGradi = angoloInRadianti π 57.2958
```

Errori

Se viene fornito un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato richiede all'utente di immettere la distanza conosciuta da un oggetto e l'altezza conosciuta dello stesso e visualizza l'angolo in gradi tra la posizione dell'utente e l'estremità superiore dell'oggetto.

```
CONST RADAGRADI = 57.2958     'costante per la conversione da radianti a gradi
INPUT "Distanza dall'oggetto: "; distanza
INPUT "Altezza dell'oggetto: "; altezza
PRINT "L'angolo in gradi tra la posizione dell'osservatore e la cima
dell'oggetto è: ";

tangenteA = altezza / distanza        'tangente dell'angolo dell'osservatore
angoloA = ATN(tangenteA)              'trova l'angolo utilizzando la funzione
                                      'arcotangente ATN
PRINT angoloA * RADAGRADI             'converte l'angolo in gradi
```

CDBL

Si consulti anche

CINT, CLNG, CSNG, FIX, INT

Scopo

La funzione CDBL converte il valore di un'espressione numerica in un numero a precisione doppia in virgola mobile.

Sintassi

```
CDBL(espr)
```

espr può essere una qualsiasi combinazione di numeri o variabili numeriche (interi, interi lunghi, a precisione semplice, a precisione doppia).

Utilizzo

```
PRINT CDBL(3 / 7)
```

Visualizza il valore .4285714285714285.

Descrizione

La funzione CDBL converte una costante numerica, una variabile numerica o un'espressione numerica in un valore a precisione doppia. Generalmente, questa funzione viene utilizzata per far eseguire a QBasic un calcolo in precisione doppia. Ad esempio, se il programma utilizza due variabili a precisione semplice, num1! e num2!, l'espressione CDBL(num1! / num2!) restituisce il risultato della divisione in precisione doppia, anziché nel valore di default, ovvero a precisione semplice.

Lo stesso risultato può essere ottenuto assegnando il risultato di un calcolo a una variabile a precisione doppia.

Ad esempio l'istruzione:

```
risultato# = num1! / num2!
```

produce un risultato a precisione doppia e memorizza il valore in risultato#.

Commenti

La funzione CDBL non può rendere più preciso un numero: anche se l'istruzione PRINT CDBL(1.1) visualizza il valore 1.100000023841858, le cifre ulteriori sono errate. Tuttavia, il discorso è differente quando si tratta di determinare i risultati di un calcolo, dato che questo (ad esempio 3 / 7) può essere eseguito con ulteriori posizioni decimali, rendendo quindi il risultato più preciso.

Nelle applicazioni pratiche e scientifiche, la precisione di un calcolo dipende dalla precisione dei componenti utilizzati per eseguirlo. Anche se è possibile dividere due numeri a precisione semplice per ottenere un risultato a precisione doppia, il risultato non è più preciso rispetto ai dati a precisione semplice utilizzati. Quindi, le ulteriori posizioni decimali trasmettono una falsa impressione di precisione; non è necessario utilizzare una precisione superiore rispetto a quella richiesta dai dati.

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato illustra il metodo per convertire il risultato di un calcolo in un valore a precisione doppia. Si noti che, assegnando il risultato di un calcolo a una variabile a precisione doppia, viene prodotto lo stesso risultato che si ottiene mediante la funzione CDBL. Inoltre, l'utilizzo separato di CDBL con il numeratore e il denominatore non offre alcun vantaggio rispetto all'utilizzo della stessa funzione con l'espressione nel suo complesso.

```
INPUT "Numeratore: "; numeratore
INPUT "Denominatore: "; denominatore

PRINT numeratore / denominatore, "(risultato a precisione semplice)"

risultato# = numeratore / denominatore
PRINT risultato#;
PRINT "(risultato assegnato ad una variabile a precisione doppia)"

PRINT CDBL(numeratore / denominatore); "(risultato convertito con CDBL)"
PRINT CDBL(numeratore) / CDBL(denominatore);
PRINT "(risultato con valori convertiti prima della divisione)"
```

CINT

Si consulti anche

FIX, INT

Scopo

La funzione CINT converte il valore di un'espressione numerica in un numero intero; ad esempio, il valore di CINT(99.7) è 100.

Sintassi

```
CINT(espr)
```

espr può essere una qualsiasi combinazione di numeri o variabili numeriche (interi, interi lunghi, a precisione semplice, a precisione doppia).

Utilizzo

```
a = 100.43
PRINT CINT(a)
```

Visualizza il valore 100.43 convertito in intero, ovvero 100.

```
PRINT CINT(-1.9)
```

Visualizza il valore -1.9 convertito in un intero, ovvero -2.

Descrizione

La funzione CINT converte i valori di un'espressione numerica in un intero mediante arrotondamento. Per i numeri positivi, le frazioni superiori a 0.5 vengono arrotondate al minimo intero superiore, mentre le frazioni inferiori a 0.5 vengono arrotondate al massimo intero inferiore. Per i numeri negativi è valido invece l'opposto: le frazioni maggiori di 0.5 vengono arrotondate al massimo intero inferiore, mentre le frazioni inferiori a 0.5 vengono arrotondate al minimo intero superiore.

CINT può essere utilizzata con qualsiasi tipo numerico, tuttavia il valore deve essere compreso nell'intervallo da -32768 a 32767, che è quello degli interi con segno.

Errori

Se il valore utilizzato con CINT è inferiore a -32768 o maggiore di 32767, QBasic visualizza il messaggio di errore: "Overflow". Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". L'utilizzo di questa funzione con un valore intero o intero lungo non provoca alcun errore, il risultato è uguale al valore fornito.

Esempio

Il programma di seguito riportato richiede di immettere un numero e quindi lo visualizza dopo averlo convertito in un intero con la funzione CINT. Per uscire dal programma, è sufficiente immettere 0 (zero).

```
DO
  INPUT "Inserire un numero (immettere 0 per uscire): "; numero
  PRINT "Il numero intero più vicino è: "; CINT(numero)
LOOP UNTIL numero = 0   'termina quando viene immesso 0
```

CLNG

Si consulti anche

CINT, FIX, INT

Scopo

La funzione CLNG converte il valore di un'espressione numerica in un numero intero lungo; viene generalmente utilizzata per convertire un valore in modo che possa

essere passato a un sottoprogramma o a una funzione che prevede un intero lungo. Le frazioni vengono arrotondate all'intero più vicino.

Sintassi

```
CLNG(espr)
```

espr può essere una qualsiasi combinazione di numeri o variabili numeriche (interi, interi lunghi, a precisione semplice, a precisione doppia).

Utilizzo

```
PRINT CLNG(-32768.92)
```

Visualizza -32769

```
PRINT CLNG(1483555.513)
```

Visualizza 1483556.

Descrizione

La funzione CLNG può essere utilizzata per convertire una variabile numerica o un'espressione in un intero lungo di 4 byte; l'eventuale parte frazionaria viene arrotondata all'intero più vicino prima della conversione. Per i numeri positivi, le frazioni superiori a 0.5 vengono arrotondate al minimo interno superiore, mentre le frazioni inferiori a 0.5 vengono arrotondate al massimo intero inferiore. Per i numeri negativi è valido invece l'opposto: le frazioni maggiori di 0.5 vengono arrotondate al massimo intero inferiore, mentre le frazioni inferiori a 0.5 vengono arrotondate al minimo intero superiore.

Per passare una variabile intera a un sottoprogramma che prevede un intero lungo, è sufficiente richiamarla con un'istruzione analoga alla seguente:

```
LungSub& CLNG(valore%)
```

Il valore utilizzato con CLNG non può essere minore di -2147483648 o maggiore di 2147483647 (l'intervallo degli interi lunghi).

Errori

Se il valore utilizzato con CLNG non è compreso nell'intervallo degli interi lunghi, QBasic visualizza il messaggio di errore: "Overflow". Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato utilizza la funzione CLNG per la chiamata a un sottoprogramma che calcola la crescita della popolazione per tre generazioni. Si noti che il valore immesso, memorizzato in numero%, è di tipo intero, mentre il sottoprogramma Popolazione prevede un intero lungo individui&. La funzione CLNG nella chiamata al sottoprogramma converte l'argomento numero% in un intero lungo, prima di passarlo a Popolazione.

```
DECLARE SUB Popolazione (individui&)
DO
  INPUT "Inserire il numero di individui: ", numero%
  IF numero% = 0 THEN EXIT DO
  Popolazione CLNG(numero%)
LOOP

SUB Popolazione (individui&)
    'mostra la popolazione dopo tre generazioni (in modo semplicistico)
    CONST FIGLIPERCOPPIA = 1.8
    FOR generazione% = 1 TO 3
        coppie& = individui& * .25    'assume che metà individui abbiano figli
        figli& = coppie& * FIGLIPERCOPPIA
        individui& = individui& + figli&
        PRINT "Popolazione dopo generazione "; generazione%; ": "; individui&
    NEXT generazione%
END SUB
```

COS

Si consulti anche

ATN, SIN, TAN

Scopo

La funzione COS restituisce il coseno di un angolo.

Sintassi

```
COS(angolo)
```

angolo è la misura dell'angolo espressa in radianti. I gradi possono essere convertiti in radianti, moltiplicando il numero di gradi per il risultato di π diviso per 180.

Utilizzo

```
PRINT COS(57.2958)
```

Visualizza il coseno di 57.2958, ovvero .7336411.

```
PRINT COS(0)
```

Visualizza il coseno di 0, ovvero 1.

```
PRINT COS(3.1415)
```

Visualizza il coseno di 3.1415, ovvero -1.

Descrizione

La funzione COS restituisce il coseno di un angolo, che deve essere espresso in radianti. Per convertire un angolo da gradi in radianti, è sufficiente utilizzare la formula radianti = gradi*pi/180, dove pi è 3.141593 in precisione semplice oppure 3.141592653589793 in precisione doppia.

La funzione restituisce per default un valore a precisione semplice, se però il valore fornito è a precisione doppia, anche il valore restituito è analogo.

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: “Mancata corrispondenza dei tipi”. Se un valore non viene convertito in radianti, prima di passarlo alla funzione COS, QBasic non visualizza alcun messaggio di errore, anche se il risultato è ogni caso errato.

Esempio

Il programma di seguito riportato stampa una tabella dei seni e coseni di 19 angoli diversi. Viene utilizzata l’istruzione PRINT USING per mostrare tutti i valori calcolati fino alla quarta posizione decimale.

```
CONST PI = 3.141593
CLS
PRINT : PRINT "Gradi   Radianti     Seno      Coseno": PRINT
FOR angolo% = 0 TO 360 STEP 20
    radianti = angolo% * PI / 180       'converte l'angolo in radianti
    PRINT USING "###        "; angolo%;
    PRINT USING "#.####     "; radianti;
    PRINT USING "##.####    "; SIN(radianti); COS(radianti)
NEXT angolo%
```

CSNG

Si consulti anche

CDBL, CINT, CLNG, FIX, INT

Scopo

La funzione CSNG converte il valore di un’espressione numerica in un numero a precisione semplice in virgola mobile; può risultare utile nel caso fosse necessario passare una variabile che non è a precisione semplice a una funzione o a un sottoprogramma che prevede questo tipo di valore.

Sintassi

```
CSNG(espr)
```

espr può essere una qualsiasi combinazione di numeri o variabili numeriche (interi, interi lunghi, a precisione semplice, a precisione doppia).

Utilizzo

```
doppiaPrec& = 1.23456789
PRINT CSNG(doppiaPrec&)
```

Visualizza il valore di doppiaPrec& convertito in precisione semplice, ovvero 1.234568.

Descrizione

La funzione CSNG converte una costante numerica, una variabile numerica o un’espressione numerica in un valore a precisione semplice in virgola mobile,

arrotondandolo al numero di posizioni decimali previste dal tipo a precisione semplice. L'utilizzo di questa funzione produce lo stesso risultato dell'assegnazione di un valore a una variabile a precisione semplice.

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Avvertenze

È necessario tenere presente che la conversione di un numero dal tipo a precisione doppia o da intero lungo in un numero a precisione semplice potrebbe diminuire l'accuratezza dei dati.

Esempio

Il programma di seguito riportato memorizza il numero immesso dall'utente in una variabile a precisione doppia, visualizza il valore in precisione doppia e quindi lo converte in precisione semplice, visualizzando il nuovo valore.

```
DO
  INPUT "Inserire un valore (immettere 0 per uscire): "; doppiaPrec#
  IF doppiaPrec# = 0 THEN EXIT DO
  PRINT "Valore a precisione doppia: "; doppiaPrec#
  PRINT "Valore a precisione singola: "; CSNG(doppiaPrec#)
LOOP
```

EXP

Si consulti anche

LOG

Scopo

La funzione EXP calcola la funzione esponenziale, ovvero il valore di e (la base dei logaritmi naturali) elevato alla potenza indicata.

Sintassi

```
EXP(espr)
```

espr può essere una qualsiasi combinazione di numeri o variabili numeriche (interi, interi lunghi, a precisione semplice, a precisione doppia).

Utilizzo

```
PRINT EXP(10)
```

Visualizza il valore di e elevato a potenza 10, ovvero 22026.46.

Descrizione

Il logaritmo naturale di un numero è la potenza a cui deve essere elevata la base e (una costante con un valore approssimativo di 2.718282) per ottenere il numero dato.

La funzione EXP, che svolge un'operazione opposta rispetto a quella della funzione LOG, restituisce il valore di e elevato alla potenza indicata.

Per default, la funzione EXP restituisce un valore a precisione semplice, se però viene fornito un valore a precisione doppia, anche il valore restituito è dello stesso tipo. Ad esempio, EXP(1#) visualizza 2.718281828459045, ovvero il valore di e in precisione doppia.

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato illustra il significato di *crescita esponenziale*, visualizzando le prime 20 potenze di e.

```
PRINT "Potenza                      e alla potenza"
FOR potenza% = 1 TO 20
    PRINT USING "##"; potenza%;
    PRINT TAB(25);
    PRINT USING "##########.####"; EXP(potenza%)
NEXT potenza%
```

FIX

Si consulti anche

CINT, INT

Scopo

La funzione FIX tronca un numero e restituisce l'intero restante; questa operazione consente di non considerare le cifre che seguono la virgola decimale e quindi di eseguire più velocemente le operazioni aritmetiche.

Sintassi

```
FIX(espr)
```

espr può essere una qualsiasi combinazione di numeri o variabili numeriche (interi, interi lunghi, a precisione semplice, a precisione doppia).

Utilizzo

```
PRINT FIX(1.5)
```

Visualizza la parte intera di 1.5, ovvero 1.

```
PRINT FIX(-1.25)
```

Visualizza parte intera di -1.25, ovvero -1.

Descrizione

La funzione FIX restituisce la parte intera della costante numerica, del letterale o dell'espressione indicati. La parte decimale, se presente, viene semplicemente troncata, senza che si verifichi alcun arrotondamento.

Le funzioni FIX e INT eseguono la stessa operazione con i valori positivi, ma nel caso di valori negativi FIX restituisce il minimo intero superiore, mentre INT restituisce il massimo intero inferiore: FIX(-0.75) restituisce 0, ma INT(-0.75) restituisce -1.

Dato che non viene eseguita alcuna operazione di arrotondamento, l'utilizzo di FIX potrebbe comportare una scarsa precisione dei dati.

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato consente di analizzare il comportamento di FIX con i valori immessi.

```
DO
  INPUT "Inserire un numero (immettere 0 per uscire): "; numero
  IF numero = 0 THEN EXIT DO
  PRINT "La parte intera di "; numero; " è "; FIX(numero)
LOOP
```

INT

Si consulti anche

CINT, FIX

Scopo

La funzione INT tronca la parte frazionaria di un numero e restituisce l'intero restante; consente di scambiare precisione con velocità dato che, ignorando le frazioni, QBasic può utilizzare l'aritmetica basata sugli interi, che è più rapida rispetto a quella basata sui valori in virgola mobile.

Sintassi

```
INT(espr)
```

espr può essere una qualsiasi combinazione di numeri o variabili numeriche (interi, interi lunghi, a precisione semplice, a precisione doppia).

Utilizzo

```
PRINT INT(3.33)
```

Visualizza la parte intera di 3.33, ovvero 3.

```
PRINT INT(-1.89)
```

Visualizza la parte intera di -1.89, ovvero -2.

Descrizione

La funzione INT restituisce la parte intera della costante numerica, del letterale o dell'espressione indicati; la parte decimale, se presente, viene semplicemente troncata, senza che si verifichi alcun arrotondamento.

Le funzioni FIX e INT eseguono la stessa operazione con i valori positivi, ma nel caso di valori negativi, FIX restituisce il minimo intero superiore, mentre INT restituisce il massimo intero inferiore: FIX(-0.75) restituisce 0, ma INT(-0.75) restituisce -1.

Dato che non viene eseguita alcuna operazione di arrotondamento, l'utilizzo di INT potrebbe comportare una scarsa precisione dei dati.

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato illustra la maggiore velocità di esecuzione delle operazioni matematiche ottenute quando i numeri vengono convertiti in interi mediante la funzione INT. Ovviamente, è necessario determinare se la perdita di precisione è accettabile per l'applicazione.

```
'realizza operazioni matematiche a virgola mobile
mobileA = 1.8975: mobileB = 2.113
tempo1 = TIMER

CLS : PRINT "Secondi trascorsi per la matematica a virgola mobile: ";
FOR tempo% = 1 TO 5000
    temp = mobileA * mobileB
NEXT tempo%
tempo2 = TIMER
PRINT tempo2 - tempo1

'realizza operazioni matematiche intere
intA% = INT(mobileA): intB% = INT(mobileB)
tempo1 =.TIMER

PRINT "Secondi trascorsi per la matemetica intera: ";
FOR tempo% = 1 TO 5000
    temp% = intA% * intB%
NEXT tempo%
tempo2 = TIMER
PRINT tempo2 - tempo1
```

LOG

Si consulti anche

EXP

Scopo

La funzione **LOG** calcola il logaritmo naturale di un numero. I logaritmi naturali non devono essere confusi con quelli comuni, che sono basati su 10 invece che sul numero **e**, il cui valore è approssimativamente 2.718282.

Sintassi

```
LOG(espr)
```

espr può essere una qualsiasi combinazione di numeri o variabili numeriche (interi, interi lunghi, a precisione semplice, a precisione doppia), ma deve essere superiore a 0.

Utilizzo

```
PRINT LOG(2.718282)
```

Visualizza il logaritmo naturale di **e**, ovvero 1.

```
PRINT LOG(100)
```

Visualizza il logaritmo naturale di **100**, ovvero 4.60517.

Descrizione

La funzione logaritmo naturale determina la potenza a cui deve essere elevata la costante **e** (2.718282 in precisione semplice) per ottenere un determinato numero. Ad esempio, dato che **e** elevato alla terza potenza è 20.08554, il logaritmo naturale di 20.08554 è 3.

Per default, la funzione **LOG** restituisce un valore a precisione semplice; se però viene fornito un valore a precisione doppia, anche il valore restituito è dello stesso tipo.

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Se viene utilizzato un valore numerico minore o uguale a 0, QBasic visualizza il messaggio di errore "Chiamata di funzione non valida".

Esempio

Il programma di seguito riportato visualizza la curva logaritmica di base 10. Viene utilizzata la funzione **Log10!** per convertire il logaritmo naturale nel logaritmo a base 10, che viene moltiplicato per un fattore di scala di 100 in modo che la progressione sia sufficientemente ampia da poter essere chiaramente visualizzata sullo schermo.

```
DECLARE FUNCTION Log10! (numero%)
SCREEN 9                'adattatore EGA o VGA
CONST maxX = 640

FOR x% = 1 TO maxX
    PSET (x%, Log10!(x%) * 100)
NEXT x%
```

```
FUNCTION Log10! (numero%)
    Log10! = LOG(numero%) / LOG(10!)
END FUNCTION
```

RANDOMIZE

Si consulti anche

RND, TIMER

Scopo

L'istruzione RANDOMIZE deve essere utilizzata per specificare un determinato valore iniziale per il generatore di numeri casuali, specificando in questo modo una serie di numeri casuali da utilizzare quando il programma richiama la funzione RND.

Se come valore iniziale viene utilizzato quello restituito dàlla funzione TIMER, RANDOMIZE genera numeri casuali imprevedibili, necessari per applicazioni quali giochi e simulazioni. La funzione TIMER restituisce il numero di secondi trascorsi dalla mezzanotte.

Sintassi

```
RANDOMIZE(espr)
```

espr può essere una qualsiasi combinazione di numeri o variabili numeriche (interi, interi lunghi, a precisione semplice, a precisione doppia).

Se non viene specificato alcun valore, RANDOMIZE richiede all'utente un valore iniziale.

Utilizzo

```
RANDOMIZE 10
```

Specifica il valore 10 come valore iniziale per il generatore di numeri casuali.

```
RANDOMIZE TIMER
```

Genera una serie di numeri casuali, utilizzando il valore restituito dalla funzione TIMER come valore iniziale.

Descrizione

L'istruzione RANDOMIZE indica un valore iniziale per il generatore di numeri casuali (la funzione RND può essere successivamente utilizzata per restituire i valori casuali dal gruppo di valori creati).

Con RANDOMIZE è possibile utilizzare qualsiasi tipo di espressione numerica; se non viene specificato nessun valore, QBasic richiede all'utente di immetterne uno compreso nell'intervallo da -32768 a 32767. Se invece nel programma non è inserita l'istruzione RANDOMIZE, tutte le chiamate alla funzione RND restituiscono i valori creati dall'utilizzo del valore iniziale 0.

Si noti che ulteriori istruzioni RANDOMIZE presenti nello stesso programma reinizializzano il generatore di numeri casuali, ma non generano l'intera sequenza dall'inizio. Se il programma crea tre numeri casuali prima di richiamare RANDOMIZE

con un nuovo valore iniziale, il quarto numero casuale è il quarto valore nella sequenza del nuovo valore iniziale.

Nel caso dei giochi, il metodo più conveniente per assicurarsi che ogni esecuzione utilizzi una serie diversa di numeri casuali consiste nell'utilizzare l'istruzione RANDOMIZE TIMER (la funzione TIMER restituisce il numero di secondi trascorsi dalla mezzanotte).

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato utilizza l'istruzione RANDOMIZE per inizializzare il generatore di numeri casuali e quindi lanciare il numero richiesto di monete; viene presa in considerazione anche la possibilità che una moneta ricada sul bordo.

```
RANDOMIZE                'richiede all'utente il seme
CLS : INPUT "Inserire il numero di monete da lanciare: ", numLanci%
teste% = 0: croci% = 0
bordi% = 0               'monete che cadono sul bordo

FOR lanci% = 1 TO numLanci%
    valCasuale = RND(1)
    IF valCasuale > .5005 THEN
       teste% = teste% + 1
    ELSEIF valCasuale < .4995 THEN
       croci% = croci% + 1
    ELSE bordi% = bordi% + 1
    END IF
NEXT lanci%

PRINT "Lanci totali: "; numLanci%
PRINT "Teste = "; teste%
PRINT "Croci = "; croci%
PRINT bordi%; " monete cadute sul loro bordo."
```

RND

Si consulti anche

RANDOMIZE, TIMER

Scopo

La funzione RND restituisce un numero casuale a precisione semplice compreso nell'intervallo da 0 a 1. La stessa sequenza di numeri casuali viene creata ogni volta che il programma viene eseguito, a meno che non venga utilizzata l'istruzione RANDOMIZE per indicarne una diversa.

Sintassi

```
RND(espr)
```

espr, che è un parametro facoltativo, può essere una qualsiasi combinazione di numeri o variabili numeriche (interi, interi lunghi, a precisione semplice, a precisione doppia); il valore di espr determina quello restituito da RND.

Utilizzo

```
PRINT RND
```

Visualizza un valore casuale.

```
PRINT RND(0)
```

Visualizza il valore dell'ultimo numero casuale.

Descrizione

La funzione RND restituisce un numero detto *casuale* che viene generato da un valore memorizzato (definito iniziale), utilizzando una formula progettata per generare numeri che non hanno modelli o ordini e sembrano quindi essere casuali. Ogni valore iniziale, tuttavia, crea una sequenza fissa di numeri; ad esempio, se il programma non contiene precedenti istruzioni RANDOMIZE, i primi tre valori restituiti da RND sono sempre .7055475, .533424 e .5795186.

L'istruzione RANDOMIZE modifica il valore iniziale e di conseguenza la sequenza generata, quindi può essere utilizzata con un valore iniziale casuale restituito dalla funzione TIMER per creare numeri casuali sotto tutti i punti di vista.

Il valore indicato nella chiamata a RND specifica quello restituito da RND: se è maggiore di 0 o non viene specificato, RND restituisce il successivo valore casuale nella sequenza; se il valore indicato è uguale a 0, RND restituisce il valore casuale restituito per ultimo; infine, se il valore è minore di 0, RND restituisce il numero casuale associato a esso.

Molte applicazioni (ad esempio le simulazioni e i giochi) richiedono numeri interi casuali che rientrano in un determinato intervallo. Per generare un intero casuale all'interno dell'intervallo inferiore - superiore, è sufficiente utilizzare la seguente espressione:

```
INT((superiore - inferiore + 1) * RND * inferiore)
```

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato definisce una funzione denominata casoInt%, che restituisce un valore calcolato da RND come intero all'interno di un intervallo determinato; questa funzione viene utilizzata per impostare le coordinate casuali x e y, il raggio e il colore, allo scopo di tracciare sullo schermo cerchi di diverse posizioni, dimensioni e colori.

Il ciclo DO...LOOP nel modulo principale continua a tracciare cerchi fino a quando l'utente non preme un tasto.

```
DECLARE FUNCTION casoInt% (piubasso!, piualto!)

SCREEN 9                  'usare 9 per EGA, 1 per CGA e 12 per VGA
maxX = 640                'usare 640 per EGA o VGA e 320 per CGA
maxY = 350                'usare 350 per EGA, 200 per CGA e 480 per VGA
colori = 15               'usare 15 per EGA o VGA e 3 per CGA
```

```
DO
   x% = casoInt(-maxX, maxX)           'coordinata casuale x
   y% = casoInt(-maxY, maxY)           'coordinata casuale y
   raggio = casoInt(5, 50)             'raggio casuale
   coloreCerchio% = casoInt(0, colori)  'colore casuale
   CIRCLE (x%, y%), raggio, (coloreCerchio%)    'disegna il cerchio
   PAINT (x%, y%), coloreCerchio%               'riempie il cerchio
LOOP WHILE INKEY$ = ""          'continua fino a quando l'utente preme un tasto


FUNCTION casoInt% (piubasso, piualto)
    'restituisce un numero casuale tra il più basso e il più alto
    casoInt = INT((piualto - piubasso + 1) * RND + piubasso)
END FUNCTION
```

SGN

Si consulti anche

ABS

Scopo

La funzione SGN restituisce il valore del segno (negativo, positivo o 0) di un numero o di un'espressione numerica; può essere utilizzata per determinare velocemente la direzione in cui si sta spostando un oggetto sullo schermo, oppure per verificare se la temperatura è sopra o sotto lo zero.

Sintassi

```
SGN(espr)
```

espr può essere una qualsiasi combinazione di numeri o variabili numeriche (interi, interi lunghi, a precisione semplice, a precisione doppia).

Utilizzo

```
PRINT SGN(-145)
```

Visualizza il segno di -145, ovvero -1.

```
PRINT SGN(0)
```

Visualizza il segno di 0, ovvero 0.

```
PRINT SGN(12645.33)
```

Visualizza il segno di 12645.33, ovvero 1.

Descrizione

La funzione SGN verifica se un valore è negativo, positivo o nullo. Ad esempio, i due test di seguito riportati sono identici.

```
numero < 0
SGN(numero) = -1
```

È necessario ricordare che SGN(0) restituisce 0, il che potrebbe non essere corretto se 0 viene considerato come valore positivo.

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato genera numeri casuali centrati su 0: circa metà di questi numeri sono positivi, mentre gli altri sono negativi. Il programma utilizza quindi SGN per individuare il segno di un numero e l'istruzione SOUND per suonare un "A" (la nota La) basso se il numero è negativo e un "A" alto se il numero è positivo (la durata delle note è casuale).

```
CONST BASSO = 220                'la nota "A"
CONST ALTO = 440                 'la nota "A" un'ottava più alta

DO
   casualeVal = RND(1) - .5              'distribuisce un valore casuale
intorno a 0
   durata = ABS(casualeVal * 50)         'specifica la durata dei toni
   IF SGN(casualeVal) = -1 THEN
      SOUND BASSO, durata
   ELSE
      SOUND ALTO, durata
   END IF
   SOUND 32767, 9.1              'circa mezzo secondo di silenzio
LOOP WHILE INKEY$ = ""
```

SIN

Si consulti anche

ATN, COS, TAN

Scopo

La funzione SIN restituisce il seno di un angolo.

Sintassi

```
SIN(angolo)
```

angolo è la misura dell'angolo, espressa in radianti. I gradi possono essere convertiti in radianti, moltiplicando il numero di gradi per il risultato di π diviso per 180.

Utilizzo

```
PRINT SIN(1)
```

Visualizza il seno di 1, ovvero 9.983342E-02.

Descrizione

La funzione SIN restituisce il seno di un angolo, che deve essere espresso in radianti. Per convertire un angolo da gradi in radianti, è sufficiente utilizzare la formula radianti

= gradi *pi/180, dove pi è 3.141593 in precisione semplice oppure 3.141592653589793 in precisione doppia.

La funzione restituisce per default un valore a precisione semplice, se però il valore fornito è a precisione doppia, anche il valore restituito è analogo.

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Se un valore non viene convertito in radianti prima di passarlo alla funzione SIN, QBasic non visualizza alcun messaggio di errore, anche se il risultato è ogni caso errato.

Esempio

Si immagini che uno scalatore lanci un rampino collegato a una fune, in modo che il gancio si fissi in alto sulla roccia. Si conosce la lunghezza della fune, mentre viene misurato l'angolo definito dalla fune e dal terreno. Il programma di seguito riportato determina l'altezza del gancio rispetto al terreno.

```
CONST PI = 3.141593
CONST GRADIARAD = PI / 180
CLS

DO
  INPUT "Numero di metri di cavo svolti (immettere 0 per uscire):", lunghezza
  IF lunghezza = 0 THEN EXIT DO
  INPUT "Angolo del cavo con la terra (in gradi): ", angolo
  radianti = angolo * GRADIARAD
  altezza = lunghezza * SIN(radianti)
  PRINT "L'altezza del gancio è "; altezza; " metri."
LOOP
```

SQR

Scopo

La funzione SQR restituisce la radice quadrata di un'espressione numerica.

Sintassi

```
SQR(espr)
```

espr può essere una qualsiasi combinazione di numeri o variabili numeriche (interi, interi lunghi, a precisione semplice, a precisione doppia).

Utilizzo

```
PRINT SQR(16)
```

Visualizza la radice quadrata di 16, ovvero 4.

```
PRINT SQR(1000)
```

Visualizza la radice quadrata di 1000, ovvero 31.62278.

Descrizione

La funzione SQR restituisce la radice quadrata di un valore numerico positivo (maggiore o uguale a 0). Il valore restituito è per default a precisione semplice, se però il valore fornito è a precisione doppia, anche il valore restituito è analogo.

Dato che QBasic non offre alcuna funzione che eleva un numero al quadrato, è necessario utilizzare a questo scopo un operatore esponenziale; il quadrato della variabile numero può essere calcolata da numero^2 o numero*numero.

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Se viene utilizzato un valore numerico inferiore a 0, QBasic visualizza il messaggio di errore: "Chiamata di funzione non valida".

Esempio

Il programma di seguito riportato utilizza il teorema di Pitagora ($c^2 = a^2 + b^2$, dove a è la lunghezza di un cateto, b è la lunghezza dell'altro e c è la lunghezza dell'ipotenusa) per calcolare la lunghezza dell'ipotenusa di un triangolo rettangolo.

```
CLS
DO
   INPUT "Lunghezza del lato adiacente (immettere 0 per uscire): "; adiacente
   IF adiacente = 0 THEN EXIT DO
   INPUT "Lunghezza del lato opposto: "; opposto
   ipotenusa = SQR(adiacente ^ 2 + opposto ^ 2)
   PRINT "La lunghezza dell'ipotenusa è: "; ipotenusa: PRINT
LOOP
```

TAN

Si consulti anche

ATN, COS, SIN

Scopo

La funzione TAN restituisce la tangente di un angolo.

Sintassi

```
TAN(angolo)
```

angolo è la misura dell'angolo, espressa in radianti. I gradi possono essere convertiti in radianti, moltiplicando il numero di gradi per il risultato di π diviso per 180.

Utilizzo

```
PRINT TAN(.75)
```

Visualizza la tangente di .75, ovvero .9315965

```
PRINT TAN(3.1415)
```

Visualizza la tangente di 3.1415, ovvero -9.265741E-05.

Descrizione

La funzione TAN restituisce la tangente di un angolo, che deve essere espresso in radianti. Per convertire un angolo da gradi in radianti, è sufficiente utilizzare la formula radianti = gradi *pi/180, dove pi è 3.141593 in precisione semplice oppure 3.141592653589793 in precisione doppia.

La funzione restituisce per default un valore a precisione semplice, se però il valore fornito è a precisione doppia, anche il valore restituito è analogo.

Errori

Se viene specificato un valore non numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Avvertenze

È bene non utilizzare la tangente di 90 gradi, che è uguale a infinito; in questo caso, QBasic non visualizza il messaggio di errore: "Overflow", ma si limita a restituire un numero molto grande.

Esempio

Il programma di seguito riportato stampa una tabella di angoli e relative tangenti.

```
CONST PI = 3.141592653589793#           'pi greco a precisione doppia
CONST GRADIARAD = PI / 180

PRINT "Angolo in Gradi        Tangente(Angolo)"

FOR angolo% = 0 TO 85 STEP 5
    PRINT USING "##"; angolo%; TAB(26);
    PRINT USING "###.######"; TAN(angolo% * GRADIARAD)
NEXT angolo%
```

Capitolo 8

Funzioni di I/O

INTRODUZIONE

In questo capitolo viene esaminata l'interfaccia tra i programmi QBasic e l'utente, comprese le istruzioni e le funzioni che consentono di eseguire le operazioni di seguito riportate.

- Generare richieste e accettare i dati immessi da tastiera.
- Verificare i tasti premuti, senza dover interrompere l'esecuzione del programma.
- Rispondere all'input dei tasti funzione, dei tasti di direzione e di combinazioni di tasti particolari.
- Visualizzare testo e valori numerici in diversi formati.
- Modificare il numero di righe e colonne di testo che possono essere visualizzate sullo schermo.
- Definire le dimensioni della finestra in cui viene visualizzato il testo.
- Cancellare lo schermo o la finestra indicata
- Controllare le dimensioni del cursore.

Nella Tabella 8.1 sono riportate in ordine alfabetico le istruzioni e le funzioni di input/output.

INPUT DA TASTIERA

QBasic offre tre metodi per la gestione dell'input da tastiera; quello standard, che consiste nell'utilizzo dell'istruzione INPUT, causa l'interruzione del programma mentre l'utente digita i caratteri, l'esecuzione non viene ripresa fino a quando l'utente non preme INVIO per indicare che l'immissione è terminata.

Tabella 8.1 Le istruzioni e le funzioni di input/output

Istruzione o funzione	Descrizione
CLS	Cancella lo schermo o la finestra specificata.
CSRLIN	Restituisce la riga attuale del cursore.
INKEY$	Legge un carattere da tastiera.
INPUT	Richiede e legge input da tastiera.
INPUT$	Legge un numero specificato di caratteri da file o da una periferica; se non viene indicato alcun file, l'input viene letto da tastiera.
LINE INPUT	Richiede e legge una stringa di caratteri da tastiera.
LOCATE	Posiziona il cursore di testo in corrispondenza della riga e della colonna indicata e specifica le dimensioni del cursore.
POS	Restituisce la colonna attuale del cursore.
PRINT	Visualizza sullo schermo testo e dati in corrispondenza della posizione del cursore.
PRINT USING	Visualizza i dati numerici e di stringa nel formato specificato.
SPC	Fa avanzare il cursore o la testina di stampa di un numero specificato di colonne rispetto all'ultimo carattere stampato.
TAB	Fa avanzare il cursore o la testina di stampa alla colonna specificata.
VIEW PRINT	Definisce l'area dello schermo in cui un programma deve dirigere tutto l'output di testo.
WIDTH	Definisce il numero di righe e di colonne sullo schermo o su un'altra periferica, oppure imposta la larghezza di un file.
WRITE	Visualizza i dati (compresi i delimitatori) sullo schermo o li scrive in un file.

QBasic dispone di due versioni dell'istruzione INPUT: la prima, l'istruzione INPUT, consente di immettere sia dati numerici che caratteri, assegnandoli quindi a una o più variabili del tipo corretto; la seconda, l'istruzione LINE INPUT, accetta i dati immessi da tastiera sotto forma di stringa di caratteri e quindi assegna tutti i caratteri a una singola variabile stringa. Il vantaggio della seconda istruzione consiste nel fatto che il programma è in grado di accettare qualsiasi tipo di dati.

Entrambe le istruzioni possono visualizzare una richiesta che illustra il tipo di input e inoltre consentono di modificare o di apportare correzioni ai dati, prima di inserirli.

Un altro metodo per gestire l'input da tastiera è quello di utilizzare la funzione INKEY$, che non interrompe il programma e si limita a verificare se un tasto è stato premuto: in questo caso, il carattere viene immediatamente restituito; altrimenti viene restituita una stringa nulla (o vuota). Questo metodo consente inoltre di gestire i tasti che non producono caratteri stampabili, ad esempio i tasti di direzione, PAGSU, PAGGIÙ e i tasti funzionali.

Esiste infine la funzione INPUT$, che legge un numero specificato di caratteri da una periferica di input, generalmente un file aperto a questo scopo. Tuttavia, se non viene specificato un file, INPUT$ legge da tastiera il numero richiesto di caratteri. Non

si consiglia di utilizzare questa funzione con la tastiera, dato che non consente di modificare l'input, non lo visualizza sullo schermo e non accetta i caratteri che non possono essere stampati; inoltre l'utente deve immettere il numero esatto di caratteri indicati, dato che INPUT$ non riconosce il tasto INVIO come segnale di fine immissione.

OUTPUT DEL TESTO

L'istruzione PRINT, che visualizza sullo schermo il testo e i valori numerici, può essere utilizzata per visualizzare il testo letterale e il contenuto delle variabili numeriche e stringa. PRINT visualizza l'output partendo dalla posizione attuale del cursore fino a quando non sono stati visualizzati tutti i caratteri o le variabili specificate.

L'istruzione WRITE, simile a PRINT, invia i dati allo schermo come se fosse una telescrivente e aggiorna la posizione del cursore dopo aver visualizzato ogni carattere, facendo scorrere i dati quando l'output raggiunge la riga inferiore dello schermo. Al contrario di PRINT, WRITE racchiude le stringhe visualizzate tra doppi apici e separa tutte le voci degli elenchi con virgole. Questa istruzione deve essere utilizzata soprattutto per inviare i dati a un file; anche quando viene sfruttata per inviare i dati allo schermo, li formatta come se dovessero essere scritti in un file sequenziale.

Formattazione della stampa

L'istruzione PRINT non formatta l'output, esiste però un'altra versione che esegue questa operazione: PRINT USING permette di specificare una stringa con degli indicatori di formato, contenente i campi in cui l'istruzione visualizza l'output. Quindi, è possibile utilizzare questa stringa di formato per stampare, ad esempio, una colonna di cifre in cui tutte le posizioni decimali sono allineate. PRINT USING risulta particolarmente utile per la progettazione di prospetti o di elenchi, in cui la chiarezza è importante.

Altre due funzioni di QBasic consentono di formattare le righe di testo. La funzione SPC inserisce un numero specificato di spazi nella riga visualizzata con PRINT, consentendo in questo modo di separare le voci stampate e di realizzare una visualizzazione chiara. TAB esegue un compito analogo, tranne per il fatto che il cursore viene spostato in corrispondenza della posizione specificata sulla riga; questa funzione consente di far rientrare il testo e di allineare le colonne di un prospetto senza alcuna difficoltà.

IL CURSORE IN MODO TESTO

Come già spiegato, le istruzioni PRINT, PRINT USING e WRITE visualizzano il testo sullo schermo in corrispondenza della posizione del cursore. L'istruzione LOCATE può essere utilizzata per portare il cursore in qualsiasi posizione sullo schermo, in modo che la stampa successiva proceda da quella posizione. Anche le funzioni TAB e SPC consentono di spostare il cursore, ma soltanto all'interno della riga dello schermo corrente.

LOCATE consente non solo di spostare il cursore in corrispondenza della riga e della colonna indicate, ma anche di modificarne le dimensioni, da invisibile al normale trattino, fino a farlo diventare un blocco.

QBasic fornisce inoltre due funzioni che restituiscono la posizione attuale del cursore: POS restituisce la colonna, mentre CSRLIN la riga.

DIMENSIONI DELLO SCHERMO

Lo schermo standard di testo visualizzato quando QBasic esegue un programma, può contenere fino a 2000 caratteri in una griglia di 25 righe per 80 colonne (nella Figura 8.1 è riportata l'organizzazione standard dello schermo). È possibile modificare questo formato utilizzando l'istruzione WIDTH.

QBasic offre soltanto due valori per la larghezza: 80 colonne (default) e 40 colonne. Utilizzando lo schermo a 40 colonne, i caratteri vengono visualizzati più grandi. Non è possibile modificare l'altezza dello schermo (il numero di righe visualizzate), a meno che il computer non disponga delle schede grafiche EGA, VGA o MCGA; il valore di default di 25 righe può essere aumentato a 43 con la scheda EGA, a 43 o 50 con la VGA. Le schede grafiche VGA e MCGA possono anche visualizzare formati di testo di 80 per 30 e di 80 per 60 caratteri, ma soltanto utilizzando le modalità grafiche 11 e 12.

L'istruzione WIDTH influisce sull'aspetto dell'output di un programma, non su quello dell'ambiente di QBasic. Anche se 80 per 25 righe è la modalità standard, è possibile modificare il default dell'ambiente avviando QBasic con il parametro /H. Ad esempio il comando:

```
QBasic /H
```

indica a QBasic di visualizzare l'ambiente utilizzando la risoluzione più alta consentita dalla scheda grafica di cui si dispone, producendo in questo modo un formato di testo di 80 per 43 caratteri con una scheda EGA e di 80 per 50 caratteri con una VGA.

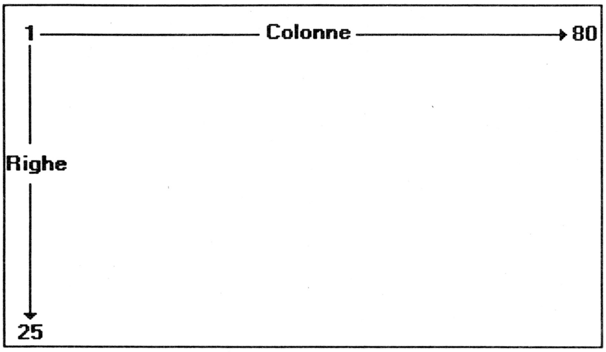

Figura 8.1 Le righe e le colonne dello schermo standard di testo

LA FINESTRA DI TESTO

QBasic consente di limitare l'output del testo soltanto a una parte dello schermo; ciò può risultare vantaggioso quando di tratta di associare una parte di testo e una di grafica sullo stesso schermo, dato che impedisce la sovrascrittura della visualizzazione grafica. L'istruzione VIEW PRINT consente di definire un numero specificato di righe dello schermo come *finestra di testo.* Dopo aver richiamato l'istruzione, le operazioni eseguite da PRINT, PRINT USING, e WRITE vengono limitate alla finestra di testo, senza toccare le aree circostanti dello schermo. L'istruzione di seguito riportata limita l'output di testo sullo schermo dalla riga 20 alla 23.

```
VIEW PRINT 20 TO 23
```

Dato che il cursore in modo testo non può essere spostato all'esterno della finestra, dopo che quest'ultima è stata impostata, l'istruzione LOCATE funziona soltanto con i numeri di riga all'interno dell'area della finestra.

CANCELLAZIONE DELLO SCHERMO

Oltre alle istruzioni che consentono di posizionare il testo sullo schermo, QBasic fornisce un'istruzione che permette di cancellare i dati inutili. Per default, CLS cancella tutto lo schermo; tuttavia, la sua azione può essere limitata in modo che venga cancellato soltanto il testo presente nella finestra di testo lasciando invariato qualsiasi grafico eventualmente presente. Oppure, se è già stata definita una finestra grafica, è possibile indicare a CLS di cancellare soltanto il contenuto di quest'ultima.

Bibliografia

Davies, Russ. *Mapping the IBM PC and PCjr.* Greensboro, N. C., Compute!Books, 1985.

Wilton, Richard. *Programmer's Guide to PC and PS/2 Video Systems.* Redmond, Wash.: Microsoft Press, 1987.

CLS

Si consulti anche

VIEW, VIEW PRINT, WIDTH (Schermo)

Scopo

L'istruzione CLS cancella la finestra attiva e riporta il cursore nell'angolo superiore sinistro della stessa. Tutte le versioni di Basic dispongono di questa istruzione, ma QBasic ne amplia le funzioni consentendo di cancellare dallo schermo soltanto la parte grafica, la parte di testo o entrambe.

Sintassi

```
CLS[0 | 1 | 2]
```

Utilizzo

```
CLS
```

Cancella la finestra attiva. Se non è definita una finestra grafica o una di testo, l'istruzione cancella tutto lo schermo.

```
CLS 0
```

Cancella tutto lo schermo, indipendentemente dalla finestra attiva.

Descrizione

Se l'istruzione CLS viene richiamata senza un argomento, non cancella tutto lo schermo di QBasic, ma soltanto la finestra attiva, che però spesso rappresenta tutto lo schermo. Tuttavia, è anche possibile suddividere lo schermo in finestre separate per la parte di testo e per quella di grafica; in questo caso, l'istruzione CLS cancella soltanto quella in fase di utilizzo.

QBasic consente inoltre di indicare la finestra da cancellare inserendo un argomento con l'istruzione CLS, infatti CLS 1 cancella la finestra grafica, CLS 2 cancella quella di testo, mentre CLS 0 cancella tutto lo schermo.

Quando viene cancellata la finestra di testo, QBasic la riempie di spazi nel colore di sfondo e riporta il cursore nell'angolo superiore sinistro della finestra. Se non è stata definita alcuna finestra di testo, QBasic cancella tutto lo schermo e riporta il cursore alla riga 1, colonna 1 (la posizione di partenza). Si noti che, quando CLS cancella una finestra di testo, non elimina la riga inferiore dello schermo, in cui sono visualizzati i tasti funzionali.

Per cancellare la finestra di testo e impostare un nuovo colore di sfondo, è sufficiente utilizzare CLS con l'istruzione COLOR. Ad esempio, le istruzioni:

```
COLOR, 1
CLS 2
```

cancellano la finestra di testo e creano un colore di sfondo blu; tutto il testo successivo viene stampato su sfondo blu, fino a quando non viene nuovamente utilizzata l'istruzione COLOR per modificarlo.

Quando l'istruzione CLS cancella la finestra grafica, ripristina anche tutti i pixel con il colore di sfondo e riporta il cursore grafico alla posizione di default, ovvero il centro della finestra. Se le nuove istruzioni grafiche utilizzano un indirizzamento relativo (mediante la parola chiave STEP), QBasic calcola la posizione del punto indicato partendo dal punto centrale della finestra.

Esempio

Il programma di seguito riportato utilizza l'istruzione CLS per assegnare a tutto lo schermo il colore di sfondo blu, quindi riempie di asterischi lo schermo, definisce una finestra di testo e utilizza CLS per cancellare soltanto quest'ultima.

```
COLOR , 1                   'imposta il colore dello sfondo a blu
CLS 0                       'riempie lo schermo con uno sfondo blu
PRINT STRING$(1920, "*");   'riempie lo schermo con asterischi
LOCATE 25, 1             'sposta il cursore sulla linea inferiore dello schermo
PRINT "Premi un tasto";     'visualizza un messaggio
SLEEP                       'attende che l'utente prema un tasto
VIEW PRINT 10 TO 20         'imposta la finestra di testo
CLS                         'ripulisce la finestra di testo
END
```

CSRLIN

Si consulti anche

LOCATE, POINT, POS, WIDTH (Schermo)

Scopo

La funzione CSRLIN restituisce il numero di riga (coordinata verticale) del cursore in modo testo (per determinare il numero di colonna del cursore è possibile utilizzare la funzione POS).

Sintassi

```
CSRLIN
```

Utilizzo

```
yPosizione% = CSRLIN
```

Assegna alla variabile yPosizione% il numero della riga dello schermo in cui è posizionato il cursore.

Descrizione

QBasic presenta due funzioni che indicano la posizione attuale del cursore. La funzione CSRLIN restituisce il numero della riga in cui è posizionato il cursore; generalmente questo numero è un intero nell'intervallo da 1 a 25, tuttavia è possibile ampliarlo fino a 43 per i sistemi che utilizzano le schede grafiche EGA e fino a 60 con le schede grafiche VGA e MCGA. La funzione POS restituisce il numero di colonna del cursore.

Esempio

I programmi che richiamano una subroutine ogniqualvolta viene attivata un'istruzione dipendente da un evento devono memorizzare la posizione del cursore in modo che possa essere ripristinata quando il controllo torna al programma chiamante. Il programma di seguito riportato utilizza l'istruzione TIMER e la funzione CSRLIN per visualizzare continuamente l'ora, mentre visualizza altre informazioni sullo schermo. L'aggiornamento dell'orologio non interferisce con l'output sullo schermo.

```
CLS : TIMER ON
ON TIMER(1) GOSUB Segno
DO
```

```
    LOCATE 2, 1: colore% = colore% + 1
    IF colore% > 7 THEN colore% = 1
    COLOR , colore%
    DO                           'colora lo schermo una linea alla volta
          PRINT " ";
          IF INKEY$ <> "" THEN   'controlla quando l'utente preme un tasto
             xCursore% = POS(0): yCursore% = CSRLIN       'trova la posizione
                                                          'del cursore
             COLOR 14, 4: LOCATE 24, 1: PRINT SPACE$(80);
             COLOR 14, 4: LOCATE 24, 1: PRINT SPACE$(80);
             LOCATE 24, 34, 1: PRINT "Uscita (S/N)? ";
             risposta$ = UCASE$(INPUT$(1))
             COLOR 7, 0: LOCATE 24, 1: PRINT SPACE$(80);
             COLOR , colore%
             IF risposta$ = "S" THEN
                fatto% = -1
             ELSE
                LOCATE yCursore%, xCursore%, 0
             END IF
          END IF
    LOOP UNTIL CSRLIN > 22
  LOOP UNTIL fatto%
  COLOR 7, 0
  END

  'questa subroutine è eseguita una volta al secondo
  Segno:
    xStop% = POS(0): yStop% = CSRLIN      'salva la posizione del cursore
    COLOR 11, 0: LOCATE 1, 36: PRINT TIME$;
    COLOR , colore%: LOCATE yStop%, xStop%    're-imposta il cursore alla sua
                                              'posizione
  RETURN
```

INKEY$

Si consulti anche

INPUT, INPUT$, LINE INPUT, ON KEY(n) GOSUB

Scopo

INKEY$ legge un carattere dalla tastiera. QBasic presenta diverse funzioni e istruzioni che consentono di accettare dati dalla tastiera, nella maggior parte dei casi però esse indicano al programma di interrompersi e attendere fino a quando l'utente non immette le informazioni. La funzione INKEY$ rappresenta un'eccezione, dato che legge un carattere da tastiera e lo restituisce immediatamente, indipendentemente dal fatto che il carattere sia disponibile; inoltre consente di gestire tasti speciali, quali quelli funzionali e di direzione.

Sintassi

INKEY$

Utilizzo

```
risposte$ = UCASE$(INKEY$)
```

Converte in lettere maiuscole un carattere letto dalla tastiera assegnandolo quindi a risposte$.

Descrizione

Quando viene premuto un tasto, il computer memorizza il carattere corrispondente in un *buffer* (un'area di memoria temporanea), fino a quando il programma non è pronto a elaborarlo. Il buffer standard può contenere un massimo di 15 caratteri. INKEY$ verifica il buffer per controllare se è già stato premuto un tasto; se contiene dei caratteri, INKEY$ legge il primo e lo restituisce al programma, in caso contrario la funzione restituisce una stringa nulla. Dato che INKEY$ non attende la pressione di un tasto, l'esecuzione del programma non viene mai interrotta.

INKEY$ restituisce una stringa di 1 o 2 byte che rappresenta il tasto premuto; per i caratteri ASCII o quelli estesi IBM, INKEY$ restituisce una stringa di 1 byte; per i tasti funzionali, una stringa di 2 byte.

Il gruppo dei tasti funzionali comprende i tasti di direzione, PAGSÙ, PAGGIÙ, HOME, FINE e i tasti da F1 a F10 (oppure da F1 a F12 se si dispone di una tastiera estesa), oltre ai tasti alfanumerici associati al tasto ALT. Quando viene premuto un tasto funzionale, INKEY$ restituisce una stringa di 2 byte consistente di un byte 0 (un carattere il cui valore ASCII è 0), seguito dal codice del tasto.

Ogni tasto ha un codice diverso, tuttavia INKEY$ legge soltanto i codici dei tasti funzionali. Le istruzioni di seguito riportate assegnano alla variabile scan%, il codice di qualsiasi tasto funzionale premuto.

```
ts$ = INKEY$
IF LEN(ts$) > 1 THEN scan% = ASC(MID$(ts$, 2, 1)
```

Nella Tabella 8.2 sono riportati i codici di tutti i tasti funzionali che INKEY$ è in grado di intercettare; l'elenco comprende soltanto i tasti che restituiscono un codice alla funzione INKEY$ di QBasic.

I tasti funzionali da F11 a F12 sono disponibili soltanto con le tastiere estese. ALT+- (trattino) e ALT-++ (più) possono essere individuati soltanto sulla tastiera principale; i tasti - e + sul tastierino numerico non restituiscono un codice, quando vengono utilizzati con ALT.

Esempio

Il programma di seguito riportato controlla la tastiera e riporta i valori ASCII e i codici relativi a ogni tasto premuto. Viene inoltre visualizzato il carattere restituito, sempre che questa operazione non disturbi la visualizzazione.

```
CLS : LOCATE 10, 1, 0
PRINT "Codice ASCII:     Carattere:      Codice:";
DO
   DO
      risposta$ = INKEY$
   LOOP WHILE risposta$ = ""              'attesa della pressione di un tasto
   codiceASCII% = ASC(risposta$)          'trova il codice ASCII
   LOCATE 10, 14: PRINT USING "###"; codiceASCII%; 'visualizza il codice ASCII
   LOCATE 10, 30
   IF codiceASCII% > 31 THEN              'visualizza i caratteri stampabili
      PRINT CHR$(codiceASCII%);
   ELSE
      PRINT " ";                          'cancella il carattere precedente
   END IF
   LOCATE 10, 45
   IF LEN(risposta$) > 1 THEN             'visualizza il codice dei tasti funzione
```

Tabella 8.2 I codici della tastiera individuati da INKEY$

Tasto	Codice	Tasto	Codice	Tasto	Codice
ALT+A	30	ALT+0	129	CTRL+F9	102
ALT+B	48	ALT++	130	CTRL+F10	103
ALT+C	46	ALT+=	131	CTRL+F11	137
ALT+D	32	F1	59	CTRL+F12	138
ALT+E	18	F2	60	ALT+F1	104
ALT+F	33	F3	61	ALT+F2	105
ALT+G	34	F4	62	ALT+F3	106
ALT+H	35	F5	63	ALT+F4	107
ALT+I	23	F6	64	ALT+F5	108
ALT+J	36	F7	65	ALT+F6	109
ALT+K	37	F8	66	ALT+F7	110
ALT+L	38	F9	67	ALT+F8	111
ALT+M	50	F10	68	ALT+F9	112
ALT+N	49	F11	133	ALT+F10	113
ALT+O	24	F12	134	ALT+F11	139
ALT+P	25	MAIUSC+F1	84	ALT+F12	140
ALT+Q	16	MAIUSC+F2	85	HOME	71
ALT+R	19	MAIUSC+F3	86	↑	72
ALT+S	31	MAIUSC+F4	87	PAGSU	73
ALT+T	20	MAIUSC+F5	88	←	75
ALT+U	22	MAIUSC+F6	89	→	77
ALT+V	47	MAIUSC+F7	90	FINE	79
ALT+W	17	MAIUSC+F8	91	↓	80
ALT+X	45	MAIUSC+F9	92	PAGSU	81
ALT+Y	21	MAIUSC+F10	93	INS	82
ALT+Z	44	MAIUSC+F11	135	CANC	83
ALT+1	120	MAIUSC+F12	136	CTRL+STAMP	114
ALT+2	121	CTRL+F1	94	CTRL+←	115
ALT+3	122	CTRL+F2	95	CTRL+→	116
ALT+4	123	CTRL+F3	96	CTRL+FINE	117
ALT+5	124	CTRL+F4	97	CTRL+PAGGIÙ	118
ALT+6	125	CTRL+F5	98	CTRL+HOME	119
ALT+7	126	CTRL+F6	99	CTRL+PAGSU	132
ALT+8	127	CTRL+F7	100		
ALT+9	128	CTRL+F8	101		

```
      codiceTastiera% = ASC(MID$(risposta$, 2, 1))
      PRINT USING "###"; codiceTastiera%;
   ELSE
      PRINT "   ";                      'cancella il carattere precedente
   END IF
LOOP UNTIL codiceASCII% = 27            'termina il ciclo se l'utente preme Esc
END
```

INPUT

Si consulti anche

INKEY$, INPUT$, LINE INPUT

Scopo

L'istruzione INPUT carica i dati da tastiera e li memorizza nelle variabili.

Sintassi

```
INPUT [;] ["prompt" {;|,}] var [, var...]
```

Il punto e virgola che segue la parola chiave INPUT mantiene il cursore sulla stessa riga, dopo che l'utente ha premuto INVIO.

prompt è una stringa facoltativa di testo che viene visualizzata subito prima del campo di immissione.

Il punto e virgola che segue il prompt provoca la visualizzazione di un punto di domanda prima del campo di immissione, mentre la virgola subito dopo il prompt elimina questo punto di domanda. Se non viene utilizzato alcun prompt, INPUT visualizza soltanto un punto di domanda.

var è la variabile di QBasic a cui viene assegnato l'input; può essere di qualsiasi tipo o precisione e può essere un elemento di un array o un tipo definito dall'utente. Per utilizzare più di una variabile, è sufficiente separarle con virgole.

Utilizzo

```
INPUT "qual è la risposta", risposta$
```

Interrompe l'esecuzione del programma e visualizza il prompt e un punto di domanda fino a quando l'utente non immette una stringa di caratteri. Quindi, la stringa viene assegnata alla variabile risposta$.

```
INPUT a%, b!, c&(d%), e$
```

Attende che l'utente immetta tre numeri e una stringa e quindi le assegna a variabili del tipo corretto.

Descrizione

L'istruzione INPUT consente di accettare i caratteri immessi da tastiera fino a quando, premendo INVIO, l'input non viene terminato; le voci dei dati immessi devono corrispondere, nel tipo e nel numero, alle variabili a cui vengono assegnate.

È possibile aggiungere un prompt per indicare all'utente il tipo di dati che il programma prevede. Se la stringa di prompt viene seguita da un punto e virgola, INPUT visualizza un punto di domanda dopo il prompt; se invece viene seguita da una virgola il punto di domanda non viene visualizzato.

INPUT visualizza sullo schermo tutti i caratteri immessi che possono essere stampati e accetta i caratteri fino a quando l'utente non preme INVIO. Questo ritorno a capo non viene considerato parte dei dati immessi, anche se generalmente il cursore si sposta all'inizio della riga successiva. L'utilizzo di un punto e virgola subito dopo

l'istruzione INPUT elimina il ritorno a capo e lascia il cursore sulla stessa riga, subito a destra rispetto all'ultimo carattere immesso.

Se INPUT richiede più di una voce, queste devono essere separate da una virgola in fase di immissione; si suppone che una nuova voce inizi con il primo carattere presente dopo la virgola e che non sia un carattere di spazio vuoto (spazio, tabulazione, avanzamento riga o ritorno a capo). L'utente può racchiudere stringhe di caratteri tra doppi apici se la stringa deve contenere spazi iniziali, due punti o virgole.

L'istruzione INPUT consente di modificare i caratteri nel campo di immissione prima di immetterli e fornisce a questo scopo un gruppo di tasti di modifica. Nella Tabella 8.3 sono riportate le combinazioni di tasti che consentono di spostare il cursore e che inseriscono o cancellano i caratteri all'interno del campo di immissione.

Errori

Immettendo troppo poche voci o dati del tipo errato non si genera alcun errore, ma QBasic visualizza il messaggio: "Ricominciare da capo"; in questo caso, l'utente deve immettere nuovamente tutte le voci nel campo di immissione. INPUT non assegna i dati alle variabili fino a quando non vengono immessi tutti correttamente.

Tabella 8.3 I tasti di modifica di INPUT

Combinazione di tasti	Descrizione
← o CTRL+]	Sposta il cursore di un carattere a sinistra.
CTRL+← o CTRL+B	Sposta il cursore di una parola a sinistra.
→ o CTRL+\	Sposta il cursore di un carattere a destra.
CTRL+→ o CTRL+F	Sposta il cursore di una parola a destra .
HOME o CTRL+K	Sposta il cursore all'inizio del campo di immissione.
FINE o CTRL+N	Sposta il cursore alla fine del campo di immissione.
INS o CTRL+R	Attiva la modalità inserimento. Quando questa modalità è attiva (indicata dal cursore a forma di blocco) e vengono inseriti i caratteri, qualsiasi carattere successivo al cursore viene spostato a destra.
TAB o CTRL+I	Inserisce una tabulazione.
CANC	Cancella il carattere in corrispondenza della posizione del cursore.
CTRL+FINE o CTRL+E	Cancella i caratteri dalla posizione del cursore fino alla fine del campo di immissione.
ESC o CTRL+U	Cancella tutto il campo di immissione.
BACKSPACE o CTRL+H	Cancella il carattere a sinistra del cursore e sposta quest'ultimo verso sinistra. Se il cursore si trova nella prima posizione del campo di immissione, viene cancellato il carattere in corrispondenza del cursore.
CTRL+T	Attiva la visualizzazione dei tasti funzionali sulla riga di stato.
INVIO o CTRL+M	Termina l'immissione e memorizza i dati contenuti nel campo.
CTRL+INTER o CTRL+C	Interrompe l'immissione e termina il programma.

Esempio

INPUT risulta maggiormente utile quando un programma deve accettare contemporaneamente diversi dati di tipo differente. Ad esempio, il programma di seguito riportato richiede di immettere un nome e la data di nascita e quindi utilizza le informazioni per calcolare il numero di giorni restanti fino al compleanno dell'utente.

```
DECLARE FUNCTION DataGiorni& (giorno%, mese%, anno&)

giorno1% = VAL(MID$(DATE$, 4, 2)):       'estrae il giorno corrente
mese1% = VAL(LEFT$(DATE$, 2)):           'estrae il mese corrente
anno1& = VAL(RIGHT$(DATE$, 4)):          'estrae l'anno corrente

CLS : PRINT
PRINT "Inserire il tuo nome e la data di nascita"
INPUT "(Nome, Giorno, Mese, Anno)>", nome$, giorno2%, mese2%, anno2&

differenza& = DataGiorni&(giorno1%, mese1%, anno1&) - DataGiorni&(giorno2%,
mese2%, anno1&)
PRINT : PRINT
IF differenza& > 0 THEN
   PRINT differenza&; "giorni dal tuo compleanno,"; nome$
   PRINT " Che regali hai avuto ?"
ELSEIF differenza& < 0 THEN
   PRINT ABS(differenza&); "giorni al tuo compleanno,"; nome$
   PRINT " Che cosa desideri ?"
ELSE
   PRINT "OGGI è il tuo compleanno, "; nome$
   PRINT " - Che bello!"
END IF
END


FUNCTION DataGiorni& (giorno%, mese%, anno&) STATIC
   'converte la data in un valore numerico
   temp& = anno& * 356 + INT((anno& - 1) \ 4) + (mese% - 1) * 28
   temp2& = VAL(MID$("000303060811131619212426", (mese% - 1) * 2 + 1, 2))
   DataGiorni& = temp& + temp2& - ((mese% > 2) AND (anno& MOD 4 = 0)) + giorno%
END FUNCTION
```

INPUT$ (tastiera)

Si consulti anche

INKEY$, INPUT, LINE INPUT

Scopo

La funzione INPUT$ (tastiera) legge un numero specificato di caratteri da tastiera; generalmente viene utilizzata per leggere i dati da un file, tuttavia, se non viene indicato alcun numero di file, INPUT$ legge l'immissione dalla periferica standard che, per default, è la tastiera.

Sintassi

```
INPUT$(caratteri)
```

caratteri è il numero di caratteri che devono essere letti e deve essere compreso nell'intervallo da 0 a 32767.

Utilizzo

```
buffer$ = INPUT$(5)
```

Legge cinque caratteri da tastiera e li assegna alla variabile stringa **buffer$**.

Descrizione

INPUT$**, in modo analogo a **INKEY$, non visualizza i caratteri ricevuti; tuttavia, è in grado di leggere più di un tasto e interrompe l'esecuzione del programma fino a quando non viene letto il numero necessario di byte. **INPUT$** non può leggere i tasti funzionali o quelli di direzione, e non consente di modificare il campo di immissione, operazione che può invece essere eseguita con **INPUT** e **LINE INPUT**.

Questa funzione gestisce i caratteri di controllo (ad esempio, le tabulazioni e il ritorno a capo) come caratteri che possono essere stampati e che vengono passati al programma, senza per questo influenzare la visualizzazione sullo schermo; l'esecuzione del programma viene interrotta fino a quando non viene letto il numero indicato di caratteri, oppure fino a quando l'utente non preme INVIO.

Nella sua forma più semplice, **INPUT$** risulta utile per effettuare una pausa nell'esecuzione del programma fino a quando l'utente è pronto a continuare, come riportato nel seguente esempio:

```
PRINT "Premere un tasto qualsiasi per continuare";
temp$ = INPUT$(1)
```

INPUT$ è inoltre utile ogniqualvolta l'utente deve inserire un numero esatto di caratteri.

Esempio

Il computer memorizza le informazioni relative ai tasti premuti in un buffer, finché il programma non è pronto a elaborarle. Questo permette all'utente di immettere le risposte prima delle richieste. Il programma di seguito riportato visualizza il contenuto di un buffer, aggiornandolo continuamente in base ai tasti premuti, utilizza **INPUT$** per svuotare il buffer, quando quest'ultimo si riempie e quando il programma termina.

```
'visualizza le intestazioni
COLOR 15, 0: CLS
READ articoli%
FOR i% = 1 TO articoli%
    READ riga%, colonna%, visualizza$
    LOCATE riga%, colonna%, 0: PRINT visualizza$;
NEXT i%
LOCATE 11, 68: COLOR 11

'imposta il segmento BIOS e visualizza l'indirizzo del buffer
DEF SEG = &H40
inizio% = &H400 + PEEK(&H80): fine% = &H400 + PEEK(&H82)
PRINT RIGHT$("0000" + HEX$(inizio%), 4); " ";
PRINT RIGHT$("0000" + HEX$(fine%), 4);

DO
```

```
        'visualizza la posizione dell'inizio del buffer (testa) e il tasto
        'successivo (coda)
        LOCATE 11, 4: COLOR 11
        testa% = &H400 + PEEK(&H1A): coda% = &H400 + PEEK(&H1C)
        PRINT RIGHT$("0000" + HEX$(testa%), 4); " ";  'visualizza la
                                                      'posizione della testa
        PRINT RIGHT$("0000" + HEX$(coda%), 4); " ";   'visualizza la
                                                      'posizione della coda
        COLOR 13: LOCATE 9, 17: PRINT SPACE$(48);
        LOCATE 9, 17 + ((testa% - &H41E) \ 2) * 3: PRINT CHR$(25); 'punta
                                                                   'alla
                                                                   'testa
        COLOR 12: LOCATE 13, 17: PRINT SPACE$(48);
        LOCATE 13, 17 + ((coda% - &H41E) \ 2) * 3: PRINT CHR$(24); 'punta
                                                                   'alla
                                                                   'coda
        'legge e visualizza i caratteri nel buffer
        FOR i% = 0 TO 15
            carattere% = PEEK(&H1E + (i% * 2))
            codiceTastiera% = PEEK(&H1E + (i% * 2) + 1)
            IF carattere% < 32 THEN tasto$ = "  " ELSE tasto$ =
            CHR$(carattere%) + " "
            LOCATE 11, 17 + (i% * 3): COLOR 14: PRINT tasto$;
'visualizza il carattere
            LOCATE 14, 17 + (i% * 3): COLOR 9
            PRINT RIGHT$("0" + HEX$(carattere%), 2);
'visualizza il codice ASCII
            LOCATE 15, 17 + (i% * 3): COLOR 10
            PRINT RIGHT$("0" + HEX$(codiceTastiera%), 2);
'visualizza il codice della tastiera
        NEXT i%

        'visualizza il numero di caratteri nel buffer
        IF testa% >= coda% THEN
           numTasti% = 16 - ((testa% - coda%) \ 2)
        ELSE
           numTasti% = (testa% - coda%) \ 2
        END IF
        LOCATE 14, 76: IF numTasti% = 16 THEN numTasti% = 0
        PRINT RIGHT$(" " + LTRIM$(RTRIM$(STR$(numTasti%))), 2);

        'se il buffer è pieno, usa INPUT$ per cancellarlo
        IF numTasti% = 15 THEN
           LOCATE 15, 67: COLOR 28: PRINT "BUFFER PIENO";
           finto$ = INPUT$(16)
           LOCATE , 67: PRINT SPACE$(11);
        END IF
LOOP UNTIL PEEK((coda% - &H400) - 2) = 27  'termina quando l'utente preme Esc

'ripristina il segmento e vuota il buffer
DEF SEG : COLOR 7, 0: LOCATE 20, 1, 1
falso$ = INPUT$(numTasti%)
END

'dati per le intestazioni delle visualizzazioni
DATA 20,6,4,"Testa Coda",6,33,"Buffer Tastiera"
DATA 6,67,"Area buffer",8,4,"041A 041C"
DATA 8,17,"1E 20 22 24 26 28 2A 2C 2E 30 32 34 36 38 3A 3C"
DATA 8,68,"0480 0482",10,3,"----|----"
DATA  10,16,"|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|"
DATA 10,67,"----|----",11,3,"         ",11,16," "
DATA 11,64," ",11,67,"         ",12,3,"----|----"
DATA  12,16,"|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|"
DATA 12,67,"----|----",14,3,"Codici ASCII"
DATA 14,67,"Attendi",15,3,"Codici tasti"
DATA 24,30,"Premi Esc per terminare"
```

LINE INPUT

Si consulti anche

INKEY$, INPUT, INPUT$, LINE INPUT #

Scopo

L'istruzione LINE INPUT accetta tutti i dati immessi come testo ASCII e li assegna a una singola variabile stringa; consente di immettere qualsiasi carattere, anche le virgole e i punti di domanda, che vengono considerati come delimitatori dall'istruzione INPUT.

Sintassi

```
LINE INPUT [;] ["prompt";] var
```

Il punto e virgola che segue la parola chiave INPUT, mantiene il cursore sulla stessa riga, dopo che l'utente ha premuto INVIO.

prompt è una stringa facoltativa di testo, visualizzata subito prima del campo di immissione, che richiede all'utente di immettere i dati richiesti.

var è la variabile di QBasic a cui viene assegnato l'input.

Utilizzo

```
LINE INPUT "Come ti chiami?"; nome$
```

Richiede di immettere un nome e assegna i caratteri immessi alla variabile stringa nome$.

Descrizione

LINE INPUT accetta tutti i caratteri che possono essere stampati e consente di modificare la riga, prima di premere INVIO. Il ritorno a capo non viene considerato parte dei dati immessi, ma termina l'immissione e sposta il cursore all'inizio della riga successiva; se invece viene inserito un punto e virgola dopo le parole chiave LINE INPUT, QBasic elimina il ritorno a capo e lascia il cursore sulla stessa riga, posizionato subito a destra rispetto all'ultimo carattere immesso.

Al contrario di INPUT, LINE INPUT non stampa un punto di domanda, a meno che non venga inserito come parte della stringa. Il campo di immissione dei dati inizia subito dopo l'ultimo carattere presente nel prompt; è bene terminare la stringa con almeno uno spazio, in modo che il prompt e l'input siano chiaramente separati sullo schermo.

La variabile a cui LINE INPUT assegna l'immissione può essere una qualsiasi variabile stringa, un elemento di un array di tipo stringa oppure una stringa a lunghezza fissa. Se l'input viene assegnato a una stringa a lunghezza fissa, LINE INPUT ignora qualsiasi carattere che supera la sua lunghezza. Se viene immesso soltanto un ritorno a capo in risposta all'istruzione LINE INPUT, questa assegna alla variabile una stringa nulla.

LINE INPUT fornisce le stesse combinazioni di tasti di modifica di INPUT.

Commenti

LINE INPUT accetta un massimo di 255 caratteri e questo numero non può essere limitato. Questo rende ovviamente questa istruzione non adatta per un input formattato, in cui l'utente deve inserire un numero fisso di caratteri (ad esempio, il numero della tessera sanitaria).

Esempio

Il programma di seguito riportato utilizza LINE INPUT per richiedere all'utente di inserire i valori separati da virgole e di memorizzarli nella variabile stringa testo$. Quindi, utilizza il sottoprogramma virgolaSepara per separare i singoli valori e inserirli in un array.

```
DECLARE SUB separaVirgola (testo$, individuale$(), totale%)

DIM individuale$(10)

CLS
LINE INPUT "Inserisci una stringa, con virgole ... "; testo$

'separa le parti individuali di testo$ e le immette nell'array individuale$()
separaVirgola testo$, individuale$(), totale%

FOR i% = 1 TO totale%
    PRINT LTRIM$(RTRIM$(individuale$(i%)))
NEXT
END


SUB separaVirgola (testo$, individuale$(), totale%) STATIC
    i% = 1
    totale% = 1

    DO
        j% = INSTR(i%, testo$, ",")

        IF j% > 0 THEN
            individuale$(totale%) = MID$(testo$, i%, j% - i%)
            totale% = totale% + 1
            i% = j% + 1
        ELSE
            individuale$(totale%) = MID$(testo$, i%)
            EXIT SUB
        END IF
    LOOP UNTIL i% > LEN(testo$)
END SUB
```

LOCATE

Si consulti anche

CSRLIN, POS, WIDTH

Scopo

L'istruzione LOCATE posiziona il cursore sullo schermo, contrassegnando in questo modo il punto in corrispondenza del quale il testo viene successivamente

visualizzato, inoltre controlla l'aspetto del cursore, rendendolo visibile o invisibile e impostandone le dimensioni all'interno della cella di carattere.

Sintassi

```
LOCATE [riga] [, [colonna] [, [visibile] [, [inizio] [, stop]]]]
```

riga è il numero di riga in cui è posizionato il cursore; generalmente è un intero nell'intervallo da 1 a 25, ma può essere esteso fino a 43 nel caso di computer che dispongono di una scheda EGA o fino a 50 con una scheda VGA. riga può trovarsi nell'intervallo da 0 a 60, con le schede VGA e MCGA nelle modalità schermo 11 e 12.

colonna è il numero di colonna in cui è posizionato il cursore ed è compreso nell'intervallo da 1 a 40 o da 1 a 80, in base alla larghezza dello schermo.

visibile determina il fatto che il cursore rimanga visualizzato o nascosto: il valore 1 visualizza il cursore, mentre 0 lo nasconde.

inizio e stop sono le righe di scansione CRT che delimitano la forma del cursore. I valori devono trovarsi nell'intervallo da 0 a 31.

Tutti gli argomenti sono facoltativi e possono essere costanti numeriche, variabili o espressioni; se ne viene omesso uno, QBasic utilizza i valori precedenti. Se viene omesso un argomento nel mezzo dell'istruzione, è necessario inserire una virgola per mantenerne la posizione. Non è indispensabile inserire le virgole nel caso di argomenti mancanti alla fine dell'istruzione.

Utilizzo

```
LOCATE, 30
```

Posiziona il cursore alla colonna 30 della riga di schermo corrente.

```
LOCATE , , 1, 0, 7
```

Rende invisibile il cursore e ne imposta le dimensioni a carattere pieno, senza modificarne la posizione.

Descrizione

L'istruzione LOCATE può essere utilizzata per posizionare il cursore di testo sullo schermo o per modificare l'aspetto del cursore.

Qualsiasi scheda crea i caratteri sullo schermo riempiendo determinate posizioni in una cella di carattere. Ad esempio, una scheda CGA utilizza una griglia di 8 per 8 per ogni cella di carattere; altre schede (comprese MDA, MCGA, EGA e VGA) utilizzano le celle di carattere con una griglia di 8 per 14, 9 per 14, 8 per 16 o 9 per 16. Se una scheda utilizza una cella di carattere con molte posizioni, può creare caratteri ad alta risoluzione.

L'istruzione LOCATE utilizza sempre una griglia di 8 per 8 per formare il cursore. Se la scheda di cui si dispone ha un carattere di cella con una risoluzione più alta, LOCATE associa i valori per le righe di inizio e di termine del cursore ai valori corretti della scheda grafica installata. Sebbene LOCATE accetti valori nell'intervallo da 0 a 31 per i valori della riga, è necessario utilizzare soltanto l'intervallo da 0 a 7, dato che l'istruzione traccia questi valori direttamente nella griglia di 8 per 8. Qualsiasi valore superiore a 7 viene convertito in un valore nell'intervallo da 0 a 7.

Se il valore della riga di termine è inferiore rispetto a quello della riga di inizio e si utilizza una scheda diversa dalla VGA (Video Graphics Array), LOCATE suddivide il cursore in due parti. Se si dispone di una scheda VGA, l'impostazione di una riga di inizio superiore a quella di termine permette di creare un cursore che riempie la cella carattere.

Avvertenze

LOCATE non sposta il cursore nella riga inferiore della visualizzazione, se è stata attivata la gestione dei tasti specificati con KEY ON (si consulti la voce KEY nel Capitolo 14). Le opzioni per l'impostazione delle dimensioni e della visibilità del cursore sono efficaci soltanto quando è attiva la modalità SCREEN 0, ovvero quella di testo. Nelle modalità grafiche, LOCATE non visualizza il cursore.

Esempio

L'istruzione LOCATE risulta utile soprattutto quando si desidera posizionare un prompt a una determinata posizione sullo schermo. Il programma di seguito riportato utilizza LOCATE per posizionare il prompt nella parte centrale dello schermo e per impostare il cursore ad altezza completa; utilizza l'istruzione INPUT quattro volte per richiedere l'immissione delle righe di testo. Prima di ogni prompt la forma del cursore viene modificata, quindi il programma visualizza le stringhe negli angoli dello schermo.

```
CLS
LOCATE 10, 1, 1, 0, 7                            'cursore ad altezza intera
INPUT "Inserire la prima stringa di testo : ", inserimento1$
LOCATE , , 1, 3, 7                               'cursore a metà altezza
INPUT "Inserire la seconda stringa di testo : ", inserimento2$
LOCATE , , 1, 5, 7                             'cursore a un quarto di altezza
INPUT "Inserire la terza stringa di testo : ", inserimento3$
LOCATE , , 1, 7, 7                               'cursore a linea singola
INPUT "Inserire la quarta stringa di testo : ", inserimento4$

'visualizza le stringhe negli angoli dello schermo
LOCATE 1, 1, 0                                   'cursore invisibile
PRINT inserimento1$
LOCATE 1, 80 - LEN(inserimento2$)
PRINT inserimento2$
LOCATE 22, 1
PRINT inserimento3$
LOCATE 22, 80 - LEN(inserimento3$)
PRINT inserimento4$
END
```

POS

Si consulti anche

CSRLIN, LOCATE, LPOS, POINT, WIDTH (Schermo).

Scopo

La funzione POS restituisce il numero di colonna (coordinata orizzontale) del cursore di testo (per restituire il numero di riga del cursore può essere utilizzata la funzione CSRLIN).

Sintassi

```
POS(n)
```

n è un argomento fittizio, sebbene non venga utilizzato dalla funzione, è sempre necessario inserirlo (si consiglia di utilizzare 0).

Utilizzo

```
xPosizione% = POS(0)
```

Restituisce il numero di colonna in cui è posizionato il cursore e assegna il valore alla variabile xPosizione%.

Descrizione

QBasic presenta due funzioni che indicano la posizione attuale del cursore. La funzione POS restituisce il numero della colonna sulla quale viene visualizzato un carattere dalla successiva istruzione PRINT. Generalmente questo valore si trova nell'intervallo da 1 a 40 o da 1 a 80, in base alla larghezza dello schermo; la colonna più a sinistra ha sempre il numero 1. La funzione CSRLIN restituisce il numero di riga del cursore.

Suggerimenti

POS risulta utile quando si desidera cancellare una riga a partire dalla posizione del cursore, infatti questa operazione può essere eseguita con la seguente istruzione:

```
PRINT SPACE$((80 - POS(0)) + 1);
```

Se non si desidera che la posizione iniziale del cursore venga modificata, è possibile salvarla in una variabile, cancellare la riga e quindi ripristinare il cursore, nel seguente modo:

```
p% = POS(0)
PRINT SPACE$((80 - p%) + 1);
LOCATE , p%
```

In modo analogo, è possibile utilizzare POS per eliminare il contenuto della riga dall'inizio fino al cursore, con le seguenti istruzioni:

```
p% = POS(0)
LOCATE , 1
PRINT SPACE$(p% - 1);
```

Esempio

Un sistema di elaborazione testi è in grado di portare a capo le parole incomplete alla fine della riga. Il programma di immissione del testo di seguito riportato emula questa caratteristica, verificando la fine della riga dopo che l'utente ha immesso ogni carattere; se individua la fine della riga durante l'immissione di una parola, quella parola viene eliminata dalla riga attuale e trasferita all'inizio di quella successiva.

```
CLS
LOCATE , , 1: PRINT TAB(33); "A CAPO AUTOMATICO"
PRINT STRING$(80, "_")
```

```
VIEW PRINT 4 TO 22
DO
   DO
       tastoPremuto$ = INKEY$
   LOOP WHILE tastoPremuto$ = ""          'aspetta che sia premuto un tasto
   codiceTasto% = ASC(tastoPremuto$)      'ottiene il valore ASCII
   IF codiceTasto% > 31 THEN              'caratteri stampabili
      buffer$ = buffer$ + tastoPremuto$
      PRINT tastoPremuto$;
      IF POS(0) > 79 THEN                 'controlla la fine della riga
         puntatore% = POS(0)              'trova l'inizio della parola
         DO
            puntatore% = puntatore% - 1
            IF MID$(buffer$, puntatore%, 1) = " " THEN EXIT DO
         LOOP UNTIL puntatore% = 1
         IF puntatore% = 1 THEN           'non sposta la riga continua
            buffer$ = "": PRINT
         ELSE                            'sposta la parola sulla riga successiva
            temp$ = MID$(buffer$, puntatore% + 1)
            LOCATE , 1: PRINT LEFT$(buffer$, puntatore%);
            PRINT SPACE$((80 - POS(0)) + 1)
            buffer$ = temp$: PRINT buffer$;
         END IF
      END IF
   ELSEIF codiceTasto% = 13 THEN        'se l'utente ha premuto Invio, inizia
                                        'una nuova riga
      buffer$ = "": PRINT
   END IF
LOOP UNTIL codiceTasto% = 27              'termina quando l'utente preme Invio
END
```

PRINT

Si consulti anche

LOCATE, PRINT # , PRINT USING, TAB, WRITE

Scopo

PRINT visualizza dati o messaggi sullo schermo.

Sintassi

```
PRINT [dati [{;|,}dati...]] [;|,]
```

dati è una qualsiasi costante numerica o stringa, variabile o espressione,; se viene specificata più di una voce, è necessario separarle con virgole o punti e virgola.

Se viene utilizzato un punto e virgola tra le voci di dati, PRINT la visualizza una di seguito all'altra; se invece viene utilizzata una virgola, PRINT visualizza la voce successiva nella zona di stampa seguente (ogni zona di stampa ha una lunghezza di 14 caratteri). La presenza di un punto e virgola o di una virgola alla fine dell'istruzione mantiene il cursore sulla stessa riga per la successiva PRINT.

Utilizzo

```
PRINT
```

Sposta il cursore di testo all'inizio della riga successiva.

```
PRINT "Premere un tasto";
```

Visualizza il messaggio posto tra doppi apici e posiziona il cursore subito dopo l'ultimo carattere stampato.

```
PRINT x, x^2, x^3
```

Visualizza il valore della variabile x, di x al quadrato, di x al cubo; il cursore di testo si sposta nella successiva zona di stampa, dopo che ogni valore è stato stampato.

Descrizione

È possibile abbreviare la parola chiave PRINT con un punto di domanda (?). Se si utilizza l'editor dell'ambiente QBasic, il punto di domanda viene tradotto in PRINT non appena viene immessa la riga.

PRINT visualizza i dati forniti sullo schermo, partendo dalla posizione attuale del cursore. Se i dati sono troppo lunghi per la parte di riga restante dopo il cursore, PRINT immette un carattere di avanzamento riga prima di inviare i dati allo schermo. Ad esempio, l'istruzione di seguito riportata sposta il cursore all'inizio della riga 11, prima di stampare 60 segni di uguale.

```
LOCATE 10, 40
PRINT STRING$(60, "=")
```

Se l'output contiene un numero di caratteri superiore rispetto a quelli che può contenere una riga, l'istruzione PRINT riempie la riga e quindi continua la visualizzazione sulla riga successiva.

Se PRINT viene utilizzato senza fornire i dati, QBasic visualizza un carattere di avanzamento riga, che sposta il cursore sulla riga successiva. Il contenuto della riga su cui era posizionato il cursore rimane invariato.

I valori numerici visualizzati da PRINT sono sempre seguiti da uno spazio; se un numero è positivo, PRINT lo fa precedere da uno spazio, se invece è negativo da un segno meno (-).

QBasic visualizza i valori numerici in un formato a virgola fissa, se possibile. Se un numero a precisione semplice può essere rappresentato con sette cifre o con un numero inferiore, viene visualizzato in questo formato, mentre i numeri composti da più di sette cifre vengono visualizzati in notazione scientifica. Ad esempio:

```
a! = 1234567: PRINT a!
```

visualizza 1234567, ma

```
b! = 12345678: PRINT b!
```

visualizza 1.234568E+07. Questo si applica anche ai numeri a precisione doppia, a parte il fatto che essi vengono visualizzati in notazione scientifica soltanto quando hanno più di 16 cifre.

Esempio

PRINT può anche visualizzare i caratteri grafici dal set di caratteri estesi IBM. In effetti, prima dell'avvento del supporto grafico ad alta risoluzione PRINT rappresentava l'unica istruzione disponibile per tracciare disegni sullo schermo. Il programma di

seguito riportato utilizza PRINT per produrre motivi caleidoscopici sullo schermo di testo.

Il programma crea i motivi sfruttando soltanto cinque caratteri:

Carattere	Codice
█	219
▄	220
▌	221
▐	222
▀	223

Si utilizza una combinazione dell'istruzione LOCATE e di PRINT per visualizzare i caratteri in linee diagonali attraverso lo schermo di testo, in sei colori diversi ad alta intensità. Dato che i motivi vengono generati in modo casuale, è possibile vedere una sequenza diversa ogni volta che viene premuto un tasto, tranne ESC, che interrompe il programma.

```
LOCATE , , 0: RANDOMIZE TIMER
sinBordo% = 1: desBordo% = 80: superiore% = 1: inferiore% = 25
DO
   COLOR RND * 7 + 8
   carattere% = RND * 5 + 218
   CLS
   cambiaX% = RND * 3 + 1: cambiaY% = SGN((RND * 2) - 2)
   x% = RND * (desBordo% - sinBordo%) + sinBordo%
   y% = RND * (inferiore% - superiore%) + superiore%
   DO
      IF x% + cambiaX% >= desBordo% OR x% + cambiaX% <= sinBordo% THEN
         cambiaX% = -cambiaX%
      END IF
      IF y% + cambiaY% >= inferiore% OR y% + cambiaY% <= superiore% THEN
         cambiaY% = -cambiaY%
      END IF
      x% = x% + cambiaX%: y% = y% + cambiaY%
      LOCATE y%, x%
      IF SCREEN(y%, x%) = carattere% THEN
         carattere% = carattere% + 1                        'cambia carattere
         IF carattere% > 223 THEN carattere% = 219
         colore% = colore% + 1: IF colore% > 15 THEN colore% = 9    'cambia
                                                                    'colore
         COLOR colore%
      END IF
      PRINT CHR$(carattere%);                               'stampa carattere
      LOCATE y%, desBordo% - x%: PRINT CHR$(carattere%);
      LOCATE inferiore% - y%, desBordo% - x%: PRINT CHR$(carattere%);
      LOCATE inferiore% - y%, x%: PRINT CHR$(carattere%);
      tastoPremuto$ = INKEY$
   LOOP WHILE tastoPremuto$ = ""
LOOP UNTIL tastoPremuto$ = CHR$(27)
END
```

PRINT USING

Si consulti anche

PRINT, PRINT # USING

Scopo

PRINT USING definisce un modello che indica la lunghezza e il formato di ogni voce da visualizzare e quindi visualizza il risultato sullo schermo. Questa istruzione deve essere utilizzata per formattare i dati nel caso in cui ogni variabile deve essere inserita in un determinato campo sullo schermo.

Sintassi

```
PRINT USING stringaformato; var[; var...] [,|;]
```

stringaformato è una costante stringa (racchiusa tra doppi apici) o una variabile, che può contenere una parte di testo, che viene stampato invariato, e uno o più caratteri di formattazione che determinano il modo in cui vengono stampate le variabili o le espressioni che seguono il modello.

Inserendo una virgola alla fine dell'istruzione, il cursore si sposta nella successiva zona di stampa sulla riga in uso, mentre il punto e virgola fa in modo che il cursore rimanga posizionato dopo l'ultimo carattere visualizzato.

Utilizzo

```
pagamento$ = "**$#####,.##"
verificaImporto! = 2449.73
PRINT USING pagamento$; verificaImporto!
```

Visualizza l'importo del pagamento in dollari con una virgola inserita ogni tre caratteri a sinistra del punto decimale. La visualizzazione viene anche riempita sulla sinistra con alcuni asterischi, in modo da creare un campo di 12 caratteri. La riga visualizzata è ***$2,449.73.

```
Cognome$: "Brunetti"
NomeCompleto$ = "Giorgio Brunetti"
PRINT USING "!.&"; NomeCompleto$; Cognome$
```

Stampa l'iniziale del nome di Giorgio Brunetti e il cognome completo, G. Brunetti. Il punto esclamativo (!) estrae il primo carattere dalla variabile NomeCompleto$; la e commerciale (&) visualizza Cognome$ senza alcuna formattazione. Si noti che, dato che il punto (.) non fa parte di un campo numerico, viene interpretato come carattere, non come punto decimale.

Descrizione

L'istruzione PRINT USING accetta qualsiasi variabile e la inserisce nei campi di una stringa modello, che controlla la lunghezza di ogni variabile, la posizione nella riga di output e la formattazione e può inoltre essere utilizzata per formattare sia i dati numerici che quelli stringa. Ogni campo nel modello viene indicato da uno o più caratteri speciali di formattazione; generalmente il loro numero determina quanti caratteri o cifre della variabile devono essere stampati. Nella Tabella 8.4 è riportato un elenco dei caratteri di formattazione riguardanti le stringhe, mentre nella Tabella 8.5 sono elencati i caratteri di formattazione relativi ai numeri.

Se a PRINT USING vengono inviati più argomenti rispetto a quelli presenti nel modello, QBasic non ignora quelli eccedenti; al contrario, dopo che il contenuto dell'ultimo campo assegnato è stato stampato, ricomincia partendo dall'inizio del

Tabella 8.4 I caratteri di formattazione di PRINT USING per le stringhe

Carattere	Descrizione
!	Visualizza soltanto il primo carattere di una stringa specificata.
&	Visualizza tutta la stringa, indipendentemente dalla lunghezza.
\\	Visualizza il numero specificato di caratteri in una stringa, determinato dal numero di spazi più le barre inverse. Se la stringa sorgente non contiene un numero di caratteri sufficiente a riempire il modello, vengono inseriti alcuni spazi a destra.
_	Indica che il carattere successivo deve essere intepretato come carattere letterale. Il trattino di sottolineatura deve essere utilizzato per inserire i caratteri di formattazione nel testo; ad esempio, "_!" stampa un punto esclamativo, non il primo carattere di una stringa.

modello, inserendo le ulteriori variabili nello stessa serie di campi. Nell'esempio di seguito riportato, le variabili a%, c% ed e% vengono formattate con il primo campo, mentre le variabili b$, d$ e f$ vengono formattate con il secondo campo.

```
PRINT USING "### \\"; a%; b%; c%; d$; e$; f$
```

Si noti in questo esempio l'importanza di non posizionare le variabili stringa in modo che vengano inserite nei campi numerici e viceversa.

Se una variabile numerica contiene un numero superiore di cifre rispetto a quelle che possono essere inserite nel campo del modello a essa assegnato, il valore visualizzato esce dal modello e viene inserito un segno di percentuale (%). PRINT USING non tronca le variabili numeriche se i relativi campi non le contengono.

Errori

Se viene specificato un valore numerico per un campo stringa o un valore stringa per un campo numerico, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Se il numero di cifre di un campo specificato nel modello di un'istruzione PRINT USING supera i 24 caratteri, QBasic visualizza il messaggio di errore: "Chiamata di funzione non valida".

Esempio

PRINT USING viene spesso utilizzata per formattare i dati, in modo che vengano visualizzati in modo corretto sullo schermo. Il programma di seguito riportato legge gli elementi delle transazioni di un cliente dalle istruzioni DATA e quindi li visualizza sullo schermo sotto forma di un'istruzione contabile. Si noti che, sebbene i dati di questo esempio vengano letti dalle istruzioni DATA, un'effettiva applicazione contabile dovrebbe ricevere i dati da un file o da tastiera.

```
CLS
dataTemp$ = " \\_/\\_/\\"                     'intestazione della data
denaroTemp$ = " L. #########."                'intestazione del denaro
```

Tabella 8.5 I caratteri di formattazione di PRINT USING per i numeri

Carattere	Descrizione
#	Visualizza una cifra numerica. Per indicare il numero massimo di cifre da visualizzare, è sufficiente utilizzare diversi simboli di numero. I numeri che hanno poche cifre vengono giustificati a destra all'interno del campo con alcuni spazi iniziali.
$$	Inserisce un segno di dollaro subito prima del numero in un campo numerico. Se il numero da visualizzare è negativo, un segno meno viene visualizzato subito a sinistra del segno di dollaro.
**	Stampa asterischi all'inizio del numero. Devono essere inseriti due asterischi, ognuno dei quali rappresenta una posizione del numero nel campo.
**$	Combina gli effetti di $$ e **, visualizzando un segno di dollaro a sinistra del numero e riempiendo con gli asterischi qualsiasi spazio iniziale prima del segno di dollaro.
+	Visualizza il segno dei numeri positivi e negativi. Se il carattere più appare all'inizio del campo, il segno viene visualizzato subito a sinistra del numero, se invece appare alla fine, il segno viene visualizzato subito a destra del numero.
,	Visualizza una virgola ogni tre cifre non decimali. Per utilizzare questo carattere, è sufficiente inserire una virgola a sinistra del punto decimale nel campo.
-	Visualizza i numeri negativi con un segno meno finale; va posto alla fine del campo.
.	Contrassegna la posizione di un punto decimale in un campo numerico. Se a sinistra del punto decimale sono specificate alcune cifre e il numero visualizzato è inferiore a 1, viene visualizzato uno 0 iniziale. La parte decimale del numero viene arrotondata per corrispondere al numero di cifre presenti a destra del punto decimale.
^^^^[^]	Stampa il numero in formato esponenziale. I quattro accenti circonflessi (^) visualizzano un esponente di due cifre (E+nn), i cinque accenti circonflessi visualizzano un esponente di tre cifre (E+nnn). Il simbolo di numero (#) deve essere utilizzato prima di questi simboli per indicare la lunghezza del campo di cifre significative. In un campo esponenziale è possibile utilizzare qualsiasi carattere di formattazione numerico, tranne la virgola.

```
transazTemp$ = " ###### \      \  \" + SPACE$(18) + "\" 'intestazione della
                                                         'transizione

READ numConto&, nomeCompleto$, cognome$
PRINT TAB(32); "SOFTWARE & ALTRO": PRINT
PRINT TAB(36); "CONTO"
PRINT USING dataTemp$; MID$(DATE$, 4, 2); LEFT$(DATE$, 2); RIGHT$(DATE$, 2)
PRINT TAB(39 - (LEN(cognome$) \ 2));
PRINT USING "!.&"; nomeCompleto$; cognome$;
PRINT TAB(62); USING "########"; numConto&
PRINT STRING$(80, "-")
PRINT "  Articolo Data    Descrizione";
PRINT TAB(49); "Credito"; TAB(70); "Debito"
```

```
PRINT STRING$(80, "-")
totale! = 0: lunghezza% = 9
DO
    READ articolo&
    IF articolo& < 0 THEN EXIT DO
    READ data$, desc$, debito!, credito!
    PRINT USING transazTemp$; articolo&; data$; desc$;
    IF debito! > 0 THEN
        PRINT TAB(65); USING denaroTemp$; debito!
        totale! = totale! + debito!
    ELSEIF credito! > 0 THEN
        PRINT TAB(45); USING denaroTemp$; credito!
        totale! = totale! - credito!
    END IF
    lunghezza% = lunghezza% + 1      'aggiunge 1 alla lunghezza della lista
LOOP
FOR i% = 1 TO 22 - lunghezza%
    PRINT
NEXT i%
PRINT STRING$(80, "-")
PRINT TAB(58); USING "Totale: L. ########,"; totale!
END
'dati del conto
DATA 460755,"Gino Rossi","ROSSI"
DATA 3850,"01/08/89","Scheda madre",2862000,0
DATA 3863,"03/08/89","Sistema",1795000,0
DATA 3864,"03/08/89","Monitor",379000,0
DATA 3879,"10/08/89","Basic",60000,0
DATA 3888,"18/08/89","Dischetti",20000,0
DATA 3925,"22/08/89","Disco fisso",529000,0
DATA 3926,"22/08/89","Assegno",0,5000000
DATA 3933,"23/08/89","Interfaccia grafica",117000,0
DATA 3976,"28/08/89","Programma di backup",149000,0
DATA -1
```

SPC

Si consulti anche

SPACE$, TAB, WIDTH (Schermo)

Scopo

La funzione SPC sposta il cursore o la testina di stampa in avanti in base al numero specificato di spazi; può essere utilizzata con le istruzioni PRINT, LPRINT e PRINT # per formattare l'output in modo che risulti facilmente leggibile.

Sintassi

```
SPC(col)
```

col è il numero di posizioni in base al quale viene avanzato il cursore o la testina di stampa. Questo valore può essere una costante, una variabile o un'espressione, ma deve essere un numero positivo nell'intervallo da 0 a 32767.

Utilizzo

```
LPRINT "CODICE"; SPC(16); "DESCRIZIONE"; SPC(19); "POSIZIONE"
```

Stampa l'intestazione di un prospetto, La stringa "CODICE" viene separata da "DESCRIZIONE" da 16 spazi; quest'ultima, a sua volta, è separata da "POSIZIONE" da altri 19 spazi.

Descrizione

La funzione SPC fa avanzare il cursore di testo o la testina di stampa di un numero specificato di colonne (partendo dalla colonna in cui è stato stampato l'ultimo carattere) e riempie le colonne restanti con spazi; può essere utilizzata per inserire un numero determinato di spazi tra le voci, indipendentemente dalla loro posizione.

SPC opera come se avesse un punto e virgola finale; se viene utilizzata alla fine di una riga di dati, viene eliminato il ritorno a capo e l'avanzamento riga, mentre il cursore o la testina di stampa vengono lasciati immediatamente a destra dell'ultimo spazio stampato.

Se la lunghezza dei dati che seguono una chiamata a SPC è superiore rispetto al numero di colonne restanti sulla riga, QBasic visualizza tutti i dati sulla riga successiva. Se il numero indicato di colonne è troppo grande per essere gestito dalla periferica di output in uso, la funzione SPC utilizza come numero di spazi il numero di colonne modulo (MOD) la larghezza effettiva disponibile.

Errori

Se viene utilizzato un valore maggiore di 32767, QBasic visualizza il messaggio di errore: "Overflow". Un valore negativo non produce un errore; QBasic si limita a ignorare la funzione.

Esempio

La funzione SPC può essere utilizzata con l'istruzione PRINT # per formattare i dati inviati a un file; eseguendo questa operazione, non è più necessario formattare nuovamente i dati quando il file viene stampato. Il programma di seguito riportato scrive un semplice motivo geometrico su un file su disco e quindi visualizza il contenuto del file.

```
CLS
OPEN "DIAGONAL" FOR OUTPUT AS #1
FOR i% = 1 TO 24
     PRINT #1, STRING$(15, CHR$(219)); SPC(i%);
     PRINT #1, STRING$(25, CHR$(219)); SPC(25 - i%);
     PRINT #1, STRING$(15, CHR$(219));
NEXT i%
CLOSE
SHELL "TYPE DIAGONAL"
END
```

TAB

Si consulti anche

SPC, WIDTH (Schermo)

Scopo

La funzione TAB imposta la posizione di colonna del carattere successivo da visualizzare sulla riga corrente; può essere utilizzata con le istruzioni PRINT, LPRINT e PRINT # quando una voce deve apparire in una particolare colonna.

Sintassi

```
TAB(col)
```

col può essere una costante, una variabile o un'espressione e indica il numero di colonna in corrispondenza della quale si sposta la testina di stampa o il cursore; deve essere un intero positivo nell'intervallo da 0 a 32767.

Utilizzo

```
PRINT #1, TAB(16) "PREZZO: "TAB(36 - (LEN(prezzo$))) prezzo$
```

Invia a un file sequenziale una riga di dati che contiene 36 caratteri: inizia con 16 spazi, prosegue con una stringa tra doppi apici e termina con il contenuto della variabile prezzo$, giustificata a destra nel campo di 36 caratteri.

Descrizione

La funzione TAB può essere utilizzata per spostare il cursore di testo o la testina di stampa in corrispondenza di un determinata colonna.

TAB non può essere utilizzata per spostarsi su un numero di colonna maggiore rispetto alla larghezza gestita dalla periferica di stampa. Il numero massimo di colonne per riga è generalmente di 80, sia per la stampante che per lo schermo; se il numero di colonna indicato è superiore rispetto alla larghezza dell'output, TAB non restituisce un messaggio di errore, ma avanza fino alla posizione corrispondente al numero di colonna modulo (MOD) la larghezza. Se TAB riceve un valore negativo, imposta il numero di colonna a 1.

Questa funzione opera come se avesse un punto e virgola finale ed elimina il normale ritorno a capo e l'avanzamento riga, generalmente aggiunti alla fine dell'istruzione PRINT; ciò consente che la posizione di stampa sia mantenuta alla colonna specificata.

Errori

Se viene utilizzato un valore maggiore di 32767, QBasic visualizza il messaggio di errore: "Overflow". Un valore negativo non produce un errore; TAB si limita a spostare il cursore di testo o la testina di stampa in corrispondenza della prima colonna della riga.

Esempio

Quando TAB viene utilizzata per formattare il testo, QBasic si assicura che la riga contenga il numero corretto di caratteri, riempiendo le colonne con spazi. Il programma di seguito riportato visualizza una pagina formattata di testo con un bordo.

```
PRINT CHR$(201); STRING$(77, CHR$(205)); CHR$(187)
FOR i% = 1 TO 5
    PRINT CHR$(186); TAB(79); CHR$(186)
NEXT
messaggio$ = "Guida di riferimento a QBasic"
PRINT CHR$(186); TAB(40 - (LEN(messaggio$) \ 2)); messaggio$; TAB(79);
CHR$(186)
FOR i% = 1 TO 4
    PRINT CHR$(186); TAB(79); CHR$(186)
NEXT
messaggio$ = "Il riferimento supremo"
PRINT CHR$(186); TAB(40 - (LEN(messaggio$) \ 2)); messaggio$; TAB(79);
CHR$(186)
messaggio$ = "per"
PRINT CHR$(186); TAB(40 - (LEN(messaggio$) \ 2)); messaggio$; TAB(79);
CHR$(186)
messaggio$ = "i programmatori QBasic"
PRINT CHR$(186); TAB(40 - (LEN(messaggio$) \ 2)); messaggio$; TAB(79);
CHR$(186)
FOR i% = 1 TO 8
    PRINT CHR$(186); TAB(79); CHR$(186)
NEXT
PRINT CHR$(200); STRING$(77, CHR$(205)); CHR$(188)
END
```

VIEW PRINT

Si consulti anche

CLS, LOCATE, PRINT, SCREEN (Funzione), VIEW (Grafici), WIDTH (Schermo)

Scopo

L'istruzione VIEW PRINT definisce un'area rettangolare dello schermo e limita tutto l'output di testo all'interno dei suoi bordi. Questa area è conosciuta come finestra di testo, o view port di testo. Se il programma utilizza sia il testo che la grafica, è possibile creare una finestra di testo che mantenga i due tipi di output separati, impedendo in questo modo che una parte di testo si sovrapponga a un disegno.

Sintassi

```
VIEW PRINT [superiore TO inferiore]
```

superiore è il numero della riga che contrassegna il limite superiore della finestra; inferiore è il numero di riga che contrassegna il limite inferiore. Entrambi i parametri hanno un intervallo di 1 fino al numero massimo di righe disponibili con la scheda grafica installata (per default, 25). Il valore della variabile inferiore deve essere maggiore rispetto a quello della variabile superiore.

Utilizzo

```
VIEW PRINT 10 TO 24
```

Definisce 15 righe dalla decima alla ventiquattresima come area per l'output del testo.

Descrizione

L'istruzione VIEW PRINT può essere utilizzata per specificare la parte dello schermo in cui viene visualizzato l'output di testo; non influenza l'output delle istruzioni grafiche.

Dopo aver definito la finestra di testo con VIEW PRINT, qualsiasi output viene visualizzato all'interno di questa finestra. Si noti che un'istruzione di output eseguita nella riga inferiore della finestra fa scorrere il testo di una riga verso l'alto, esattamente come se fosse l'effettiva riga inferiore dello schermo.

Se la visualizzazione dei tasti funzionali è attiva, non è possibili indicare la riga inferiore dello schermo come parametro di VIEW PRINT.

Il numero di righe disponibili per la finestra dipende da quello che l'adattatore video di cui si dispone è in grado di visualizzare. Il numero è generalmente 25, tuttavia, se il sistema dispone di un adattatore EGA è possibile ampliare tale numero fino a 43 (50 con l'adattatore VGA e 60 con l'adattatore VGA o MCGA nelle modalità 11 e 12).

Errori

Dopo aver definito la finestra di testo, non è possibile spostare il cursore all'esterno dell'area. Ad esempio, se vengono utilizzate le istruzioni:

```
VIEW PRINT 1 TO 12
LOCATE 16, 1
```

QBasic visualizza il messaggio di errore: "Chiamata di funzione non valida". QBasic visualizza lo stesso messaggio di errore se viene utilizzato un numero negativo, se il valore nella riga superiore è maggiore rispetto al valore della riga inferiore, oppure se il valore della riga inferiore supera il numero di righe disponibili.

Esempio

VIEW PRINT viene generalmente utilizzata nelle applicazioni che comportano l'utilizzo di testo e di grafica. La definizione di finestre separate per queste parti permette di isolare le due forme di output, impedendo che interferiscano l'una con l'altra. Il programma di seguito riportato crea un grafico a torta nella finestra grafica in base ai dati immessi nella finestra di testo.

```
CONST PI = 3.14159

'dimensioni degli array ed etichette dei grafici a torta
DIM articoli(1 TO 7) AS SINGLE, etichetta(1 TO 6) AS STRING

'imposta la modalità schermo e il viewport testo
SCREEN 9: LINE (0, 0)-(639, 349), 1, BF
VIEW PRINT 20 TO 25
CLS 2: PRINT "Immettere fino a sei valori per il grafico a torta ";
PRINT "(0 = ultimo articolo, -1 = uscita)": PRINT STRING$(80, "_")
DO

    'raccoglie i dati per i grafici a torta nel viewport testo (max sei articoli)
    VIEW PRINT 22 TO 25
    CLS 2                       'ripulisce il viewport testo
    totale! = 0
    titolo$ = LEFT$(titolo$, 60)
    FOR numArticoli% = 1 TO 6
```

```
        PRINT "Valore per l'articolo #"; numArticoli%; : INPUT ;
        articoli(numArticoli%)
        totale! = totale! + articoli(numArticoli%)
        IF articoli(numArticoli%) = 0 THEN
           PRINT
           INPUT "Titolo del grafico :", titolo$
           EXIT FOR
        ELSEIF articoli(numArticoli%) < 0 THEN EXIT FOR
        END IF
        PRINT TAB(41); "Etichetta"; : INPUT etichetta(numArticoli%)
        NEXT numArticoli%
        numArticoli% = numArticoli% - 1
        IF articoli(numArticoli% + 1) < 0 THEN EXIT DO

        VIEW PRINT               'disabilita il viewport testo
        'inizializza il viewport grafico
        VIEW (10, 5)-(629, 245), 0, 15
        CLS 1                    'ripulisce il viewport grafico
        inizio! = .0001: unita! = 2 * PI / totale!
        riga% = 2: colonna% = 12
        'disegna il diagramma a torta
        FOR i% = 1 TO numArticoli%
            fine! = articoli(i%) * unita! + inizio!
            IF fine! > 2 * PI THEN fine! = 2 * PI
            CIRCLE (310, 120), 80, i%, -inizio!, -fine!
            centro! = fine! - ((fine! - inizio!) / 2)
            x = 310 + 40 * COS(centro!)
            y = 120 - 40 * SIN(centro!)
            PAINT (x, y), i%, i%
            CIRCLE (310, 120), 80, 0, -inizio!, -fine!

            'disegna la legenda e stampa le etichette
            inizio! = fine!
            LINE (10, colonna%)-(20, colonna% + 5), i%, BF
            LOCATE riga%, 10: PRINT LEFT$(etichetta(i%), 20)
            riga% = riga% + 1: colonna% = colonna% + 14
        NEXT i%
        LOCATE 17, 40 - (LEN(titolo$) \ 2): PRINT titolo$;
LOOP WHILE 1
END
```

WIDTH (Schermo)

Si consulti anche

SCREEN (Istruzione), WIDTH (File), WIDTH LPRINT

Scopo

L'istruzione WIDTH ripristina il numero di colonne sullo schermo. Se il computer dispone di un adattatore video ad alta risoluzione, è anche possibile sfruttare questa istruzione per aumentare il numero di righe disponibili sullo schermo.

Sintassi

```
WIDTH [colonne] [, righe]
```

colonne è il numero di caratteri presenti su una riga dello schermo, 40 o 80 (il valore di default). righe è il numero di righe dello schermo, ovvero 25 (il valore di default), 43, 30, 50 o 60.

Utilizzo

```
WIDTH 40
```

Imposta il numero di colonne a 40.

```
WIDTH, 43
```

Imposta il numero di righe a 43, senza modificare il numero di colonne.

Descrizione

L'istruzione WIDTH deve essere utilizzata per selezionare il numero di colonne e di righe di testo che possono essere visualizzate. La modifica delle dimensioni della visualizzazione cancella il contenuto dello schermo.

I valori massimi per il numero di righe dipendono dalla modalità attivata per lo schermo e dalle capacità dell'adattatore video installato, mentre il numero di colonne è sempre 40 o 80. Nella Tabella 8.6 sono riportate le dimensioni disponibili in ogni modalità video.

WIDTH risulta particolarmente utile nelle modalità video da 9 a 13, in cui è possibile utilizzarla per controllare l'altezza delle etichette dei grafici.

Tabella 8.6 I valori validi per WIDTH nelle modalità video di QBasic

Modalità video	Risoluzione testo	Adattatori
0	80 per 25	MDA, Hercules, Olivetti, AT&T, CGA, EGA, MCGA, VGA
	80 per 43	EGA, VGA
	80 per 50	VGA
	40 per 25	CGA, EGA, MCGA, VGA
	40 per 43	EGA, VGA
	40 per 50	VGA
1	40 per 25	CGA, EGA, MCGA, VGA
2	80 per 25	CGA, EGA, MCGA, VGA
3	80 per 25	Hercules
4	40 per 25	Olivetti e AT&T
7	40 per 25	EGA, VGA
8	80 per 25	EGA, VGA
9	80 per 25	EGA, VGA
	80 per 43	EGA, VGA
10	80 per 25	EGA, VGA
	80 per 43	EGA, VGA
11	80 per 30	MCGA, VGA
	80 per 60	MCGA, VGA
12	80 per 30	VGA
	80 per 60	VGA
13	40 per 25	MCGA, VGA

É necessario utilizzare WIDTH per impostare le dimensioni dei caratteri, prima di iniziare a disegnare; in caso contrario, qualsiasi grafico esistente viene eliminato dallo schermo. È inoltre necessario tenere presente che le modalità video da 9 a 12 non consentono di utilizzare meno di 80 colonne.

Errori

Se si cerca di impostare il numero di colonne a un valore diverso da 40 o 80, oppure il numero di righe a un valore che non è supportato dall'adattatore, QBasic visualizza il messaggio di errore: "Chiamata di funzione non valida".

Esempio

Il programma di seguito riportato illustra la differenza nelle dimensioni dei caratteri quando la larghezza dello schermo viene modificata. Viene utilizzata l'istruzione WIDTH per modificare da 80 a 40 il numero di colonne in una riga.

```
CONST MESSAGE$ = "Guida di riferimento a QBasic"
CLS
WIDTH 80, 25
PRINT MESSAGE$
LOCATE 23, 1
PRINT "Premi un tasto per cambiare la larghezza dello schermo a 40 colonne"

DO
LOOP WHILE INKEY$ = ""

WIDTH 40, 25
PRINT MESSAGE$
```

WRITE

Si consulti anche

PRINT, WRITE #

Scopo

L'istruzione WRITE visualizza i dati sullo schermo, separando ogni voce con una virgola e racchiudendo ogni stringa tra doppi apici.

Sintassi

```
WRITE [espr]
```

espr è un elenco di una o più voci di dati, separate da virgole; queste voci possono essere numeriche o variabili stringa, costanti o espressioni. Se non viene specificato un elenco di espressioni, vengono scritti soltanto un carattere di ritorno a capo e di avanzamento riga.

Utilizzo

```
WRITE
```

Porta il cursore alla riga successiva.

```
a% = 6: prezzo = 18000: tassa = 1.19
WRITE a%, "Frullatore", prezzo * tassa
```

Visualizza tre voci di dati sullo schermo, partendo dalla posizione del cursore. WRITE visualizza delle virgole per separare le voci di output e cancella tutti gli spazi.

Descrizione

L'istruzione WRITE può essere utilizzata per inviare i dati allo schermo, partendo dalla posizione attuale del cursore e facendo avanzare quest'ultimo su una nuova riga, dopo che i dati sono stati scritti. QBasic valuta le espressioni numeriche presenti nell'elenco dei dati e WRITE visualizza il valore risultante, senza spazi iniziali e finali, racchiudendo le stringhe tra doppi apici e separando le voci dell'elenco con virgole. Le virgole presenti nell'elenco delle espressioni non spostano il cursore nella successiva zona di stampa, operazione eseguita invece da PRINT. In effetti, con WRITE le virgole vengono utilizzate come separatori tra le voci di dati.

Il carattere di avanzamento riga può essere eliminato aggiungendo un punto e virgola, come nella funzione PRINT.

Esempio

Il programma di seguito riportato confronta WRITE e PRINT, utilizzando le due istruzioni per visualizzare in modi diversi gli stessi dati sullo schermo.

```
CLS
DIM parola(1 TO 9) AS STRING

FOR i% = 1 TO 9
    READ parola(i%)              'legge una voce dei dati
    PRINT i%, parola(i%),        'visualizza la voce con PRINT
    WRITE i%, parola(i%)         'visualizza la voce con WRITE
NEXT i%
PRINT

'visualizza il dato sullo schermo con PRINT
PRINT parola(1); parola(2); parola(3); parola(4); parola(5); parola(6);
parola(7); parola(8); parola(9)


'visualizza il dato sullo schermo con WRITE
WRITE parola(1), parola(2), parola(3), parola(4), parola(5), parola(6),
parola(7), parola(8), parola(9)
END

DATA "La ","scrittura ","è stata ","fondamentale ","per ","la civiltà, "
DATA "forse ","l'invenzione ","più grande."
```

Capitolo 9

Gestione degli errori

INTRODUZIONE

Gli errori che si verificano nei programmi possono essere suddivisi in due categorie principali: gli errori di compilazione e quelli di esecuzione. I primi comprendono gli errori di sintassi (ad esempio un parola chiave di QBasic scritta in modo errato) e i riferimenti alle variabili array che utilizzano un indice all'esterno dell'intervallo di definizione. QBasic avvisa immediatamente l'utente di tali errori, quando si cerca di eseguire un programma nell'ambiente di sviluppo; inoltre se i programmi vengono creati utilizzando questo ambiente, QBasic avvisa l'utente della presenza di errori di sintassi in una riga subito dopo l'immissione.

Gli errori di esecuzione sono più difficili da individuare. Per definizione, essi si verificano soltanto quando il programma viene eseguito, dopo che la compilazione è già stata effettuata correttamente. Senza dimenticare che un particolare errore potrebbe anche non verificarsi ogniqualvolta il programma viene eseguito, ma apparire soltanto quando si verifica una determinata combinazione di circostanze.

Un errore che si verifica durante l'esecuzione potrebbe anche non essere responsabilità del programmatore; ad esempio, se si dimentica di accendere la stampante prima che il programma provi a stampare un prospetto, si verifica un errore di esecuzione. Questo è un tipo di errore che il programma potrebbe essere in grado di gestire, intercettandolo oppure terminando correttamente in modo che l'utente non perda alcun dato.

In questo capitolo viene esaminato il modo in cui QBasic consente di intercettare e di gestire gli errori di esecuzione. Nella Tabella 9.1 sono riportate in ordine alfabetico le istruzioni e le funzioni di gestione degli errori offerte da QBasic.

Tabella 9.1 Le istruzioni e le funzioni di gestione deegli errori di QBasic

Istruzione o funzione	Descrizione
ERDEV	Restituisce il codice di errore DOS impostato dal gestore degli errori e le informazioni riguardanti il dispositivo in cui si è verificato l'errore.
ERDEV$	Restituisce il nome del dispositivo che ha generato l'errore.
ERL	Restituisce il numero di riga dell'istruzione che ha provocato l'errore.
ERR	Restituisce il codice QBasic relativo all'errore più recente.
ERROR	Simula un errore.
ON ERROR GOTO	Stabilisce l'intercettazione degli errori e specifica una routine di gestione degli errori.
RESUME	Trasferisce il controllo dalla routine di gestione degli errori all'istruzione che ha provocato l'errore.

INTERCETTAZIONE DEGLI ERRORI

Molti errori di esecuzione vengono innanzitutto individuati dall'hardware del computer. Se ad esempio il programma indica alla CPU di dividere un numero per 0, la CPU invia un interrupt al DOS; in modo analogo, se il programma cerca di leggere un disco quando lo sportello dell'unità è aperto, il DOS genera un interrupt di errore, che produce il familiare messaggio:

```
Annulla, Riprova, Tralascia ?
```

In ogni modo, QBasic individua gli interrupt generati dagli errori e li passa alle proprie routine interne di gestione degli errori; inoltre, dopo aver raccolto le informazioni relative alle circostanze che hanno provocato l'errore, le passa al programma sotto forma di un messaggio in una finestra di dialogo.

Queste informazioni possono aiutare a comprendere in che modo vada modificato il programma per impedire che tale errore si verifichi nuovamente. Tuttavia, il messaggio di errore non impedisce che l'errore interrompa il programma; per gestire tutti gli errori è necessario utilizzare l'istruzione ON ERROR GOTO, che attiva l'intercettazione degli errori e indica il numero di riga o di etichetta che contrassegna l'inizio di una routine di gestione degli errori nel programma. Si noti che, se viene utilizzata l'istruzione ON ERROR GOTO in un sottoprogramma o in una funzione, il numero di riga o l'etichetta che l'istruzione indica devono trovarsi nel programma principale, non nel sottoprogramma o nella funzione.

Se viene utilizzata un'istruzione ON ERROR GOTO, il programma non viene più arrestato da un errore, ma l'esecuzione passa invece alla routine specificata, che contiene le istruzioni necessarie la gestione degli errori.

GESTORI DEGLI ERRORI

Se il programma individua degli errori, è necessario creare una routine di gestione degli errori, definita *gestore degli errori*, che indichi le operazioni da eseguire quando si verifica un determinato errore. QBasic fornisce funzioni che restituiscono informazioni relative alla natura di ogni errore; queste funzioni possono essere utilizzate nel gestore di errori, in modo che quest'ultimo sia in grado di rispondere in modo corretto. La funzione ERR restituisce il codice QBasic dell'errore, la funzione ERDEV restituisce il codice DOS dell'errore e le informazioni relative al dispositivo in cui si è verificato un errore hardware, mentre la funzione ERDEV$ restituisce il nome del dispositivo. Se il programma utilizza i numeri di riga, la funzione ERL restituisce il numero della riga in cui si è verificato l'errore.

Dopo ogni errore QBasic disabilita l'intercettazione degli errori, fino a quando non termina la routine di gestione. Questo impedisce di eseguire il gestore degli errori in modo ricorsivo, se si verifica un altro errore. Tuttavia, se durante l'esecuzione della routine di gestione si verifica un errore, quest'ultimo non viene intercettato e quindi il programma termina e visualizza un messaggio di errore.

Il programma può incontrare due tipi di errori, quelli "riparabili" e quelli "fatali". Nel primo caso, la routine di gestione degli errori può intraprendere le operazioni necessarie a correggerli (ad esempio, indicando all'utente di accendere la stampante). Dopo aver offerto una soluzione al problema, il programma può utilizzare l'istruzione RESUME per cercare di eseguire nuovamente l'istruzione che ha provocato l'errore, oppure può ignorarlo, utilizzando RESUME NEXT per eseguire l'istruzione successiva a quella che ha provocato l'errore. Se invece l'errore è fatale e non può essere corretto o ignorato, il programma deve richiamare l'istruzione RESUME con un numero di riga o un'etichetta per trasferire il controllo a una routine di uscita che consenta di terminare in modo corretto l'esecuzione.

Se si verifica un errore che la routine non è in grado di gestire, è possibile passare il controllo a QBasic, utilizzando l'istruzione di seguito riportata, all'interno di una routine di gestione degli errori.

```
ON ERROR GOTO 0
```

Questa istruzione fa in modo che QBasic termini il programma e stampi il proprio messaggio di errore; utilizzandola all'esterno di una routine di gestione degli errori è possibile disattivare l'intercettazione degli errori, senza per questo terminare il programma.

QBasic fornisce inoltre un'istruzione che consente di verificare le routine di gestione degli errori nel programma che si sta sviluppando. L'istruzione ERROR simula un determinato errore, in modo da non dover attendere che questo si verifichi naturalmente; quando viene richiamata, il programma esegue la routine di gestione degli errori indicata dall'ultima istruzione ON ERROR GOTO, proprio come se si fosse verificato un errore.

GESTIONE DEGLI EVENTI

QBasic consente anche di intercettare altri eventi che potrebbero verificarsi durante l'esecuzione del programma. Questi eventi sono predefiniti e hanno istruzioni dedicate; ad esempio, ON COM GOSUB trasferisce l'esecuzione a una speciale routine ogniqualvolta il programma riceve dati da una porta seriale; ON TIMER GOSUB consente di creare una subroutine che viene eseguita ogni volta che sono trascorsi un numero specificato di secondi in base all'orologio di sistema.

bibliografia

Duncan, Ray, et al. *The MS-DOS Encyclopedia.* Redmond, Wash.: Microsoft Press, 1988.

The Waite Group. *MS-DOS Papers.* Indianapolis, Ind.: Howard W. Sams & Co., 1988.

ERDEV

Si consulti anche

ERDEV$, ERR, IOCTL, IOCTL$, ON ERROR GOTO

Scopo

La funzione ERDEV restituisce un intero che descrive il dispositivo che ha prodotto un errore e l'errore stesso; può essere richiamata per determinare il tipo di errore che si è verificato. Il programma può così intraprendere le misure corrette per rettificare il problema oppure per portare a termine l'esecuzione in modo da non perdere alcun dato.

Sintassi

```
ERDEV
```

Utilizzo

```
CodiceErrore% = ERDEV
```

Assegna il codice di errore alla variabile di tipo intero CodiceErrore%, dopo che si è verificato un errore hardware.

Descrizione

Il valore intero restituito da ERDEV contiene due parti separate di informazioni. Il byte più basso, quello meno significativo (LSB ovvero Least-Significant Byte) contiene il codice di errore del DOS che rivela la natura dell'errore. Il byte più alto, quello più significativo (MSB, ovvero Most-Significant byte) contiene invece le informazioni dalla *parola attributo del dispositivo*, un insieme di flag di 2 byte che fa

parte del driver del dispositivo in memoria. Per separare le due componenti delle informazioni restituite da ERDEV, è sufficiente utilizzare le seguenti istruzioni:

```
perifericaID% = VAL("&H" + LEFT$(HEX$(ERDEV), 2))
CodiceErrore% = VAL("&H" + RIGHT$(HEX$(ERDEV), 2))
```

Si noti che la traduzione nel e dal formato esadecimale maschera il bit del segno, impedendo in questo modo che perifericaID% e CodiceErrore% diventino numeri negativi. Per quanto riguarda l'interpretazione delle informazioni contenute in perifericaID% (il MSB del valore restituito da ERDEV), si consulti la Tabella 9.2. I valori possibili per CodiceErrore% (il LSB del valore restituito da ERDEV) sono invece riportati nella Tabella 9.3.

Tabella 9.2 Il significato dei bit nel MSB restituito da ERDEV

Bit	Significato se il bit è uguale a 1
0	Il dispositivo è quello standard di input
1	Il dispositivo è quello standard di output
2	Il dispositivo è il dispositivo NUL
3	Il dispositivo è l'orologio
4	Non utilizzato (sempre 0)
5	Il dispositivo è un'unità disco non IBM
6	Il dispositivo supporta il controllo di I/O
7	Il dispositivo è un dispositivo di carattere

Tabella 9.3 I codici di errore nel LSB restituito da ERDEV

Codice di errore	Descrizione
0	Violazione di una protezione di scrittura
1	Unità sconosciuta
2	Dispositivo non pronto
3	Comando sconosciuto
4	Errore nei dati (CRC)
5	Lunghezza errata della struttura della richiesta al drive
6	Errore di ricerca
7	Tipo di supporto sconosciuto
8	Settore non trovato
9	Fine della carta nella stampante
10	Errore di scrittura
11	Errore di lettura
12	Errore generico
13	Non utilizzato
14	Non utilizzato
15	Sostituzione del disco non valida (soltanto DOS 3.x)

Avvertenze

QBasic conserva le informazioni impostate dall'ultimo errore di periferica, persino dopo che è stata richiamata la funzione ERDEV, è quindi possibile chiamare questa funzione più di una volta per lo stesso errore. Tuttavia, è necessario assicurarsi che il programma non legga un errore precedente, rispondendo quindi a un errore che ormai non esiste più. Se un errore non è stato causato da un dispositivo, ERDEV continua a restituire le informazioni relative al dispositivo che ha provocato per ultimo un errore. A questo punto potrebbe essere più utile fare affidamento sulla funzione ERR, che restituisce sempre il codice dell'errore attuale.

Esempio

ERDEV risulta maggiormente utile nei programmi che devono comunicare con i dispositivi installati dall'utente. Tuttavia, è anche possibile sfruttarla per raccogliere le informazioni relative ai didpositivi standard, ad esempio le unità disco e le stampanti. Il programma di seguito riportato utilizza ERDEV in un gestore degli errori per indicare il significato del codice di errore; quando viene eseguito, è opportuno lasciare aperto lo sportello dell'unità A, in modo che l'istruzione OPEN generi un errore che possa essere rilevato.

```
ON ERROR GOTO Gestore

OPEN "A:\ERDEV.DAT" FOR OUTPUT AS #1
END

Gestore:
    codice% = VAL("&H" + RIGHT$(HEX$(ERDEV), 2))
    SELECT CASE codice%
        CASE 0
            PRINT "Violazione di una protezione da scrittura"
        CASE 1
            PRINT "Unità sconosciuta"
        CASE 2
            PRINT "Dispositivo non disponibile"
        CASE 3
            PRINT "Comando sconosciuto"
        CASE 4
            PRINT "Errore nei dati (CRC)"
        CASE 5
            PRINT "Lunghezza errata della struttura della richiesta al drive"
        CASE 6
            PRINT "Errore di ricerca"
        CASE 7
            PRINT "Tipo di supporto sconosciuto"
        CASE 8
            PRINT "Settore non trovato"
        CASE 9
            PRINT "Fine della carta nella stampante"
        CASE 10
            PRINT "Errore di scrittura"
        CASE 11
            PRINT "Errore di lettura"
        CASE 12
            PRINT "Errore generico"
    END SELECT
    RESUME NEXT
```

ERDEV$

Si consulti anche

ERDEV, ERR, IOCTL, IOCTL$, ON ERROR GOTO

Scopo

La funzione ERDEV restituisce il nome del dispositivo che ha generato un errore; può essere richiamata per permettere al programma di raccogliere le informazioni che consentono di rettificare il problema oppure di terminare in modo da non perdere alcun dato.

Sintassi

```
ERDEV$
```

Utilizzo

```
periferica$ = ERDEV$
```

Assegna alla variabile periferica$, il nome del dispositivo in cui si è verificato l'errore.

```
IF ERDEV$ = "LPT1 "THEN PRINT "Errore provocato dalla stampante"
```

Stampa un messaggio se la stampante collegata alla porta parallela ha generato un errore.

Descrizione

La funzione ERDEV$ restituisce il nome del dispositivo che ha provocato un errore. Se un dispositivo di blocco, ad esempio un'unità disco (A: o B:) ha generato l'errore, questa funzione restituisce una stringa di due caratteri; altrimenti restituisce una stringa di otto caratteri che contiene il nome del dispositivo, come la porta della stampante o la porta seriale (ad esempio, LPT1 o COM1). Talvolta l'errore è provocato da uno dei dispositivi elencati nel file CONFIG.SYS; in questo caso la funzione ERDEV$ restituisce il suo nome.

Avvertenze

QBasic conserva le informazioni impostate dall'ultimo errore di periferica persino dopo che è stata chiamata la funzione ERDEV$, è quindi possibile chiamare questa funzione più di una volta per lo stesso errore. Tuttavia, è necessario assicurarsi che il programma non legga un errore precedente, rispondendo quindi a un errore che ormai non esiste più. Se un errore non è stato causato da un dispositivo, ERDEV$ continua a restituire le informazioni relative al dispositivo che ha provocato per ultimo un errore.

Esempio

La funzione ERDEV$ risulta utile soprattutto nei programmi che devono interfacciarsi con dispositivi installati dall'utente. Si esamini l'esempio riportato nella voce ON ERROR GOTO per verificare una routine di controllo degli errori che utilizza

ERDEV$ per consentire all'utente di decidere in che modo correggere un errore riparabile.

ERL

Si consulti anche

ERDEV, ERDEV$, ERR, ERROR, ON ERROR GOTO, RESUME

Scopo

La funzione ERL restituisce il numero di riga dell'istruzione che ha provocato un errore e può essere utilizzata soprattutto come strumento per la messa a punto e per la correzione degli errori di esecuzione che si verificano nel programma.

Sintassi

```
ERL
```

Utilizzo

```
PRINT ERL
```

Visualizza il numero di riga dell'istruzione che ha provocato l'errore.

```
Rigaerrore& = ERL
```

Assegna a Rigaerrore& il numero di riga in corrispondenza della quale si è verificato l'ultimo errore.

Descrizione

ERL risulta utile soltanto se il programma utilizza i numeri di riga, mentre non funziona con le etichette di riga e non restituisce neanche il numero di riga del codice sorgente contenente l'errore. Se la riga in cui si è verificato l'errore non contiene un numero di riga, ERL restituisce il numero dell'ultima riga prima di quella su cui si è verificato l'errore. Il valore che ERL restituisce è un numero di riga nell'intervallo da 0 a 65529; dato che QBasic considera ogni numero di riga superiore a 65529 come etichetta di riga, ERL non restituisce un valore superiore a 65529.

Esempio

Dato che l'utente di un programma non può generalmente accedere al codice sorgente, la funzione ERL rappresenta uno strumento per la messa punto, utilizzata durante lo sviluppo di un programma e quindi cancellata al termine dell'applicazione. La routine di debug qui di seguito riportata, che deve essere inserita nella versione di test di un programma, indica il tipo di errore che si è verificato e il relativo numero di riga (si noti che il programma restituisce il numero di riga 100, ovvero quello precedente all'istruzione che ha provocato l'errore).

```
1 ON ERROR GOTO Debug

10              'qui inizia il codice del programma

100


ERROR 73        'sottopone a test il gestore degli errori generando un errore

9999 END

Debug:
    PRINT : PRINT : PRINT
    PRINT "Si è appena verificato un errore di tipo "; STR$(ERR);
    PRINT " alla linea "; STR$(ERL): BEEP: PRINT
    ON ERROR GOTO 0
```

ERR

Si consulti anche

ERDEV, ERDEV$, ERL, ERROR, ON ERROR GOTO, RESUME

Scopo

La funzione ERR restituisce il codice di errore QBasic dell'ultimo errore verificatosi. Questo codice può essere utilizzato in una routine di gestione degli errori per identificare il problema e decidere se il programma può continuare.

Sintassi

```
ERR
```

Utilizzo

```
PRINT ERR
```

Visualizza il codice di errore QBasic dell'ultimo errore verificatosi.

```
CodiceErr% = ERR
```

Assegna il valore dell'errore più recente alla variabile di tipo intero CodiceErr%.

```
ERROR ERR
```

Ripete l'ultimo errore verificatosi. Se questa istruzione viene utilizzata all'interno di un gestore degli errori, interrompe il programma e stampa il messaggio di default relativo all'errore.

Descrizione

La funzione ERR restituisce un valore compresonell'intervallo da 1 a 255, che rappresenta il codice di errore QBasic dell'ultimo errore verificatosi (si consulti l'Appendice A per un elenco completo dei codici di errore di QBasic e del loro significato). Se non si è verificato alcun errore, ERR restituisce 0.

QBasic utilizza soltanto i numeri da 1 a 76 per indicare i diversi errori. Se l'errore non rientra in una categoria predefinita, QBasic restituisce il codice di errore 21 e visualizza il messaggio: "Errore non visualizzabile".

Se viene simulato un errore mediante l'istruzione ERROR, ERR restituisce il codice corrispondente all'errore simulato. ERR può essere chiamata da un punto qualsiasi all'interno del programma.

Esempio

La funzione ERR può essere utilizzata per verificare un errore, in modo da controllare se può essere corretto dall'utente. La routine di gestione degli errori di seguito riportata identifica quattro errori a cui è possibile porre rimedio e indica all'utente le operazioni corrette da eseguire.

```
DIM errMessaggio(0 TO 4) AS STRING

errMessaggio(0) = "027061068071"        'valori dei codici d'errore
errMessaggio(1) = "Carta della stampante esaurita, iinserirla"
errMessaggio(2) = "Disco pieno, inserire un nuovo disco"
errMessaggio(3) = "La periferica non è in linea, stabilire il collegamento"
errMessaggio(4) = "Il drive non è pronto, inserire correttamente il disco"

CLS
ON ERROR GOTO Intercettazione
ERROR 61     'simula un errore di disco pieno e verifica il funzionamento
             'della routine

Fine:
    COLOR 7, 0
END

Intercettazione:
        xPos% = POS(0): yPos% = CSRLIN: COLOR 14, 4
        LOCATE 23, 1: PRINT SPACE$(160);
        codiceErr$ = RIGHT$("00" + LTRIM$(RTRIM$(STR$(ERR))), 3)
        recuperabile% = INSTR(errMessaggio(0), codiceErr$)
        LOCATE 23, 6

        IF recuperabile% THEN
            recuperabile% = recuperabile% \ 3 + 1
            PRINT LTRIM$(RTRIM$(ERDEV$));
            PRINT errMessaggio$(recuperabile%); " e riprovare.": BEEP
            PRINT "   Premi un tasto quando pronti, Esc per terminare. > ";
            LOCATE , , 1
            r$ = INPUT$(1)
            IF r$ = CHR$(27) THEN RESUME Fine  'termina se l'utente preme Esc
            COLOR 7, 0: LOCATE 23, 1: PRINT SPACE$(160);
            LOCATE yPos%, xPos%, 0: RESUME
        ELSE
            PRINT "Si è verificato un errore di tipo "; STR$(ERR); "!"
            BEEP: PRINT "      "; : ERROR ERR
        END IF
```

ERROR

Si consulti anche

ERDEV, ERDEV$, ERL, ERR, ON ERROR GOTO, RESUME

Scopo

L'istruzione **ERROR** simula il verificarsi di un determinato errore e risulta particolarmente utile nelle routine di gestione degli errori di debug. Invece di attendere il verificarsi di un errore, è possibile simularlo per controllare se il gestore risponde in modo corretto. L'istruzione **ERROR** può anche essere utilizzata per definire propri codici di errore non indicati nei codici predefiniti di QBasic.

Sintassi

```
ERROR codiceErrore
```

codiceErrore è il numero dell'errore che si desidera simulare e deve essere compreso nell'intervallo da 1 a 255.

Utilizzo

```
ERROR 10
```

Genera l'errore: "Duplice definizione".

```
ERROR 251
```

Genera un errore definito dall'utente (QBasic non definisce il codice di errore 251).

```
ERROR ERR
```

Ripete l'ultimo errore (se questa istruzione viene utilizzata all'interno di un gestore degli errori, interrompe il programma e stampa il messaggio di default relativo a questo errore).

Descrizione

L'istruzione **ERROR** fa in modo che il programma si comporti esattamente come se si fosse verificato un errore. Se è abilitata l'intercettazione degli errori, il programma passa alla riga indicata dall'ultima istruzione **ON ERROR GOTO**; in caso contrario, QBasic interrompe l'esecuzione e visualizza il messaggio di errore corrispondente al codice di errore indicato.

Se il valore indicato corrisponde al numero utilizzato da QBasic e l'intercettazione degli errori non è attivata, il programma si interrompe e genera il messaggio di errore corretto; se il valore non corrisponde a un numero, il programma si interrompe e visualizza il messaggio di errore: "Errore non visualizzabile". Questo messaggio può essere tuttavia sostituito con uno proprio, creando una routine di gestione degli errori che rilevi tale errore.

L'istruzione **ERROR** può essere utilizzata per definire errori personalizzati, in modo che sia possibile verificare l'occorenza di circostanze o eccezioni che si desidera prevenire, ma che non è facile provocare. Per utilizzare questa tecnica è necessario assicurarsi che il codice di errore prescelto non sia uno di quelli riservati di QBasic; quindi, è possibile verificarlo nella routine di gestione degli errori.

Errori

Se in un'istruzione ERROR viene specificato un valore superiore a 255 o inferiore a 1, QBasic visualizza il messaggio di errore: "Chiamata di funzione non valida".

Esempio

La maggior parte dei programmi pone dei limiti sul tipo di voci che un utente può inserire. Ad esempio, un programma potrebbe rifiutare dati non corretti, anche se si trovano all'interno dei limiti della variabile e generalmente non provocherebbero un errore di input. Il gioco di seguito riportato risponde agli errori provocati dall'utente, utilizzando ERROR per definire un insieme di errori definiti dal programma.

```
RANDOMIZE TIMER
CLS
PRINT "L'obiettivo di questo gioco è di evitare che l'uomo cada fuori"
PRINT "dalla tavola. Usa i tasti freccia sinistra e destra per aiutarlo."
PRINT "< Premi un tasto per continuare >"
DO WHILE INKEY$ = ""
LOOP

CLS : LOCATE 12, 25, 0: PRINT STRING$(30, CHR$(223))
x% = 40: contatore% = 0: punteggio = 0: vite% = 5
sinFreccia$ = CHR$(0) + CHR$(75)
desFreccia$ = CHR$(0) + CHR$(77)
esc$ = CHR$(27)
ON ERROR GOTO Intercettazione
DO
   LOCATE 1, 38: PRINT USING "#####"; punteggio%
   LOCATE 11, x%: PRINT " ";
   IF contatore% > 20 THEN
       x% = x% + SGN(RND - .5) * (livelloDiff% / 200 + 1)
       punteggio% = punteggio% + 1
       livelloDiff% = livelloDiff% + 1         'rende il compito più difficile
       contatore% = 0
   END IF
   tastoPremuto$ = INKEY$
   IF tastoPremuto$ = sinFreccia$ THEN x% = x% - 1
   IF tastoPremuto$ = desFreccia$ THEN x% = x% + 1
   IF tastoPremuto$ = esc$ THEN EXIT DO
   LOCATE 11, x%: PRINT CHR$(2);
   IF x% < 25 THEN ERROR 251    'errore per caduta sul lato sinistro
   IF x% > 55 THEN ERROR 252    'errore per caduta sul lato destro
   contatore% = contatore% + 1
LOOP WHILE vite%
LOCATE 24, 35: PRINT "Hai perso";
END

Intercettazione:
    LOCATE 11, x%: PRINT " "; : livelloDiff% = 0: x% = 40
    SELECT CASE ERR
    CASE 251  'errore definito dal programma per la caduta sul lato sinistro
        LOCATE 24, 25: PRINT "Sei caduto oltre il lato sinistro!";
        BEEP: LOCATE 24, 25: PRINT SPACE$(35);
        vite% = vite% - 1
        RESUME NEXT
    CASE 252  'errore definito dal programma per la caduta sul lato destro
        LOCATE 24, 25: PRINT "Sei caduto oltre il lato destro!";
        BEEP: LOCATE 24, 25: PRINT SPACE$(34);
        vite% = vite% - 1
        RESUME NEXT
    CASE ELSE
        LOCATE 24, 26: ERROR ERR
    END SELECT
```

ON ERROR GOTO

Si consulti anche

ERDEV, ERDEV$, ERL, ERR, ERROR, RESUME

Scopo

L'istruzione ON ERROR GOTO abilita la gestione degli errori e indica la posizione del gestore degli errori; consente di rilevare gli errori di esecuzione che generalmente arrestano l'esecuzione, in modo che il programma esegua un gestore degli errori che indichi l'errore e consenta all'utente di porvi rimedio.

Sintassi

```
ON ERROR GOTO [riga | etichetta]
```

riga o etichetta è il numero di riga o l'etichetta che indica la prima riga di programma del gestore degli errori.

Utilizzo

```
ON ERROR GOTO Trappola
```

Attiva l'intercettazione degli errori e indica Trappola come etichetta della routine di gestione degli errori.

```
ON ERROR GOTO 0
```

Disattiva la gestione degli errori.

Descrizione

Il programma deve eseguire un'istruzione ON ERROR GOTO, prima di essere in grado di rilevare gli errori. È possibile utilizzare diverse istruzioni ON ERROR GOTO, in base alle routine di cui si dispone che sono in grado di gestirle. Ogniqualvolta il programma esegue una nuova istruzione ON ERROR GOTO, sostituisce le informazioni dall'istruzione precedente con le nuove.

La sezione del codice specificata da ON ERROR GOTO viene denominata gestore degli errori o routine di gestione degli errori.

Dopo che il programma ha eseguito un'istruzione ON ERROR GOTO, qualsiasi errore successivo fa in modo che il controllo passi al numero di riga o all'etichetta specificato. Quindi, il programma esegue una routine di gestione degli errori, fino a quando non incontra un'istruzione RESUME, che restituisce il controllo alla funzione interrotta o passa a un'altra parte del programma.

Se viene specificato il numero di riga 0 in un'istruzione ON ERROR GOTO, quest'ultima disattiva la gestione degli errori. Se il programma si trova già in un gestore degli errori, quando viene specificato 0 si interrompe e stampa il messaggio di errore corretto.

Errori

Se in un'istruzione ON ERROR GOTO viene specificato un numero di riga o un'etichetta che non è presente nel programma, QBasic visualizza il messaggio di errore: "Etichetta non definita".

Esempio

Molti errori che si verificano durante l'esecuzione possono essere corretti senza per questo dover terminare il programma, se quest'ultimo indica all'utente in che modo risolvere il problema. Il programma di seguito riportato utilizza l'istruzione ON ERROR GOTO in modo che il controllo passi alla routine di gestione degli errori che riconosce i due errori a cui è possibile porre rimedio, consentendo all'utente di risolvere il problema.

```
CONST FALSO = 0, VERO = NOT FALSO

ON ERROR GOTO Intercettazione  'avvia l'intercettazione degli errori e
                               'specifica il gestore degli errori

CLS
ERROR 27        'simula un errore di fine carta della stampante per
verificare che la routine lavori correttamente

Uscita:
    COLOR 7, 0: LOCATE 20, 1, 1: RESET
END

Intercettazione:
    periferica$ = ERDEV$'ottiene il nome della periferica che ha causato
l'errore
    recuperabile% = FALSO
    COLOR 14, 4

    SELECT CASE ERR
    CASE 71
        messaggio1$ = "LA STAMPANTE " + periferica$ + " HA ESAURITO LA CARTA"
        messaggio2$ = "Inserire la carta e premere un tasto."
        recuperabile% = VERO
    CASE 72
        messaggio1$ = "IL DISCO NEL DRIVE " + periferica$ + " NON E' PRONTO"
        messaggio2$ = "Controllare che sia chiuso correttamente e premere un
        tasto."
        recuperabile% = VERO
    CASE ELSE
        messaggio1$ = "Si è verificato un errore di tipo" + STR$(ERR) + "."
    END SELECT

    IF NOT recuperabile% THEN
        LOCATE 24, 1: PRINT SPACE$(80);
        messaggio1$ = messaggio1$ + "Il programma è abortito. Premere un
        tasto."
        LOCATE 24, 6: PRINT messaggio1$; : BEEP
        r$ = INPUT$(1)
        RESUME Uscita
    ELSE
        colonna% = POS(0): riga% = CSRLIN
        LOCATE 23, 1: PRINT SPACE$(160);
        LOCATE 23, 40 - (LEN(messaggio1$) \ 2): PRINT messaggio1$;
        messaggio2$ = messaggio2$ + " Premere Esc per terminare il
        programma."
        LOCATE 23, 40 - (LEN(messaggio2$) \ 2): PRINT messaggio2$;
```

```
        r$ = INPUT$(1)
        IF r$ = CHR$(27) THEN RESUME Uscita
        COLOR 7, 0: LOCATE 23, 1: PRINT SPACE$(160);
        LOCATE riga%, colonna%
        RESUME
    END IF
```

RESUME

Si consulti anche

ON ERROR GOTO

Scopo

L'istruzione RESUME re-indirizzare l'esecuzione del programma dal gestore degli errori. Se un errore di esecuzione può essere risolto, la routine di gestione degli errori che lo individua, deve consentire che l'esecuzione del programma prosegua fino a quando non è stato corretto.

Sintassi

```
RESUME [riga | etichetta]
RESUME NEXT
```

riga o etichetta è il numero di riga o l'etichetta che indica la riga in corrispondenza della quale riprende l'esecuzione.

Utilizzo

```
RESUME
```

Prova nuovamente a eseguire l'istruzione che ha provocato l'errore.

```
RESUME NEXT
```

Salta all'istruzione che ha provocato l'errore e riprende l'esecuzione dall'istruzione successiva.

```
RESUME Uscita
```

Riprende l'esecuzione, dopo un errore, con l'istruzione successiva all'etichetta Uscita.

Descrizione

L'istruzione RESUME restituisce il controllo all'istruzione che ha provocato l'errore, per consentire al programma di riprendere l'esecuzione. RESUME NEXT salta l'istruzione che ha causato l'errore e trasferisce il controllo all'istruzione successiva. RESUME con un numero di riga o di etichetta specificati passa l'esecuzione alla riga indicata.

Questa istruzione può essere utilizzata soltanto in una routine di gestione degli errori che sia già stata abilitata dall'istruzione ON ERROR GOTO. Quando un errore di esecuzione è la causa immediata di un salto alla routine di gestione degli errori, QBasic disattiva la rilevazione degli errori successivi; questo impedisce di eseguire in

modo ricorsivo il gestore degli errori, nel caso si verificasse un altro errore. RESUME consente nuovamente l'intercettazione degli errori, in modo che il gestore degli errori specificato nella precedente istruzione ON ERROR GOTO venga di nuovo utilizzato.

Tutte le routine di gestione degli errori devono contenere almeno un'istruzione RESUME per trasferire nuovamente il controllo al programma principale. Il programma non può essere concluso all'interno del gestore.

RESUME risulta utile quando è possibile porre rimedio a un errore, ad esempio quando il programma cerca di leggere da un'unità disco mentre lo sportello è aperto. Se il gestore degli errori è abbastanza sofisticato da riconoscere questo errore, è in grado innanzitutto di indicare all'utente di chiudere lo sportello dell'unità e quindi di eseguire un'istruzione RESUME per provare nuovamente l'operazione.

Se non è possibile riprovare, ma l'errore non era sufficientemente grave da richiedere l'interruzione del programma, il gestore può eseguire l'istruzione RESUME NEXT per ignorare la riga contenente l'errore e continuare l'elaborazione con l'istruzione successiva.

È bene non utilizzare istruzioni RESUME che indicano un numero di riga o un'etichetta, infatti il codice alla riga di destinazione deve essere in grado di gestire i ritorni dagli errori in tutto il programma. Specificare una riga può essere utile per due ragioni principali: gestire i ritorni da intercettazioni di errori speciali realizzate in riferimento a una sezione del programma particolarmente delicata e trasferire il controllo alla routine che termina il programma correttamente, quando tutto il resto non produce alcun risultato. La riga di destinazione deve trovarsi nel programma principale, non è possibile utilizzare RESUME con un numero di riga o un'etichetta per passare a un sottoprogramma o a una funzione. Tuttavia, se si è verificato un errore in un sottoprogramma o in una funzione, l'istruzione RESUME senza un argomento oppure l'istruzione RESUME NEXT restituiscono il controllo al sottoprogramma o alla funzione stessa.

Errori

Se una routine di gestione degli errori non comprende un'istruzione RESUME, QBasic visualizza il messaggio di errore: "Manca RESUME". Se RESUME viene utilizzata con un numero di riga o di etichetta che non esiste, QBasic visualizza il messaggio di errore: "Etichetta non definita".

Esempio

Un programma può essere eseguito con diversi tipi di configurazione del computer, determinando innanzitutto le capacità del sistema al momento dell'esecuzione e quindi modificando il proprio comportamento di conseguenza. Il metodo più semplice per eseguire questa operazione consiste nell'utilizzare una caratteristica specifica dell'hardware e controllare se si verifica un errore. La routine di seguito riportata utilizza questa tecnica per verificare se il sistema supporta una visualizzazione di 43 righe; se il sistema ne supporta soltanto 25, il programma utilizza l'istruzione RESUME NEXT per continuare.

```
lineeVideo% = 43                    'ipotizza uno schermo di 43 linee
ON ERROR GOTO VideoIntercet         'abilita l'intercettazione degli errori
WIDTH , 43                          'tenta di impostare 43 linee
ON ERROR GOTO 0                     'disabilita l'intercettazione degli errori

Uscita:
   IF lineeVideo% = 43 THEN WIDTH , 25  'reimposta la schermo standard
END

VideoIntercet:
    lineeVideo% = 25                         'cambia in 25 linee
    IF ERR = 5 THEN                          'controlla se l'errore era una
                                             'chiamata di funzione non valida
       RESUME NEXT                           'continua l'esecuzione
    ELSE
       RESUME Uscita
    END IF
```

Capitolo 10

Data e ora

INTRODUZIONE

QBasic offre diverse istruzioni e funzioni che gestiscono la data e l'ora, consentendo in questo modo di fornire agli utenti informazioni riguardanti la data e la durata di un determinato avvenimento. Il compito principale svolto da queste istruzioni riguarda la visualizzazione della data e dell'ora. Altre caratteristiche di QBasic permettono invece di progettare un'attività tenendo conto del tempo necessario o di eseguire un compito a intervalli regolari.

Nella maggior parte dei computer, l'ora viene registrata da un microchip; sui PC IBM, il circuito temporizzatore (il timer Intel 8253-5) genera un'istruzione di interruzione (interrupt) 18.2 volte per secondo. Durante ogni interruzione, definita *ciclo di clock*, il BIOS prende nota che si è verificato un altro ciclo. Quando si richiede di visualizzare l'ora, il DOS la calcola contando il numero di cicli trascorsi a partire dalla mezzanotte.

In questo capitolo vengono esaminate le capacità offerte da queste istruzioni e funzioni di QBasic e vengono presentati alcuni suggerimenti relativi al loro utilizzo. Nella Tabella 10.1 sono riportate in ordine alfabetico le relative istruzioni e funzioni.

IMPOSTAZIONE E VISUALIZZAZIONE DELLA DATA

Quando il computer viene avviato, il DOS in certi casi richiede di indicare la data e l'ora, anche se la maggior parte dei computer ormai dispone di orologi che non si fermano quando il computer viene spento. Questi orologi mantengono la data e l'ora attuali, che vengono sfruttate dal DOS per attribuire la data e l'ora di creazione ai file su disco, ovvero le informazioni che vengono visualizzate quando si utilizza il comando DIR per scorrere l'elenco della directory che contiene i file creati. In QBasic è possibile utilizzare l'istruzione DATE$ per modificare la data utilizzata dal DOS.

Tabella 10.1 Le istruzioni e le funzioni relative alla data e all'ora di QBasic

Istruzione o funzione	Descrizione
DATE$ (Istruzione e funzione)	Imposta o restituisce la data attuale.
ON TIMER GOSUB	Indica la posizione di una routine di gestione degli eventi del timer e il numero di secondi trascorsi tra le esecuzioni della routine.
SLEEP	Sospende il programma.
TIME$ (Istruzione e funzione)	Imposta o restituisce l'ora attuale.
TIMER	Restituisce il numero di secondi trascorsi dalla mezzanotte.
TIMER ON/OFF/STOP	Attiva, disattiva o sospende temporaneamente la gestione degli eventi del timer indicata dall'istruzione ON TIMER.

È necessario tenere presente che, se si utilizza questa istruzione per modificare la data, ai file creati o aggiornati viene attribuita una data errata.

Per accedere alla data attuale in QBasic, è sufficiente utilizzare DATE$ come funzione che restituisce la data odierna. Tuttavia non si deve dimenticare che, se è stata utilizzata l'istruzione DATE$ per modificare la data, la funzione DATE$ restituisce la nuova data impostata.

IMPOSTAZIONE E VISUALIZZAZIONE DELL'ORA

Oltre alla data, il DOS attribuisce anche l'ora di creazione o di aggiornamento del file. L'istruzione TIME$ consente di modificare l'ora utilizzata dal DOS; quando viene utilizzata, il DOS tiene traccia dell'ora partendo dal momento indicato. Se l'ora viene modificata utilizzando TIME$, a qualsiasi file creato o aggiornato viene attribuita l'ora in base a quella specificata nell'istruzione.

Per accedere all'ora in QBasic, è sufficiente utilizzare la funzione TIME$, che restituisce l'ora attuale. Tuttavia, se l'istruzione TIME$ è stata utilizzata per modificare l'ora, la funzione TIME$ restituisce la nuova ora impostata. È necessario tenere presente che i computer non sono dei cronometri perfetti e quindi gli orologi interni sono leggermente imprecisi. Dato che il timer dei PC IBM esegue un ciclo 18.2 volte al secondo, l'ora ha una precisione approssimativa di 54/1000 di secondo.

Se è necessario registrare il tempo necessario a eseguire una determinata operazione, è possibile utilizzare la funzione TIMER, che restituisce il numero di secondi passati dalla mezzanotte: è sufficiente richiamare TIMER, eseguire l'operazione e quindi richiamare nuovamente la funzione. La differenza tra i due risultati è il numero di secondi necessari per portare a termine l'operazione.

Un programma che utilizza TIMER può presentare un problema potenziale, nel caso un utente lo esegua a notte tarda. Dato che questa funzione restituisce il numero di secondi trascorsi dalla mezzanotte, TIMER restituisce il valore 0 a mezzanotte; ne

consegue che, se un programma è in fase di esecuzione alle 23.59 e si interrompe alle 24.01, è necessario modificare il risultato ottenuto, come illustrato nel seguente esempio:

```
oraInizio& = TIMER
.
.
.
oraFine& = TIMER

IF oraFine& < oraInizio& THEN  'il controllo è stato effettuato a mezzanotte
   oraFine& = oraFine& + 86400 'modificare l'ora
END IF
PRINT oraFine& - oraInizio&; "secondi trascorsi"
```

SOSPENSIONE DEL PROGRAMMA

Se si vuole fornire all'utente la possibilità di leggere una schermata o di eseguire un'operazione, è possibile sfruttare l'istruzione SLEEP, che sospende il programma in base al tempo specificato. Gli utenti che hanno terminato la lettura della schermata o completato l'operazione prima del tempo indicato, possono semplicemente premere INVIO per interrompere l'istruzione SLEEP.

IMPOSTAZIONE E GESTIONE DEGLI EVENTI DEL TIMER

QBasic fornisce inoltre una serie di istruzioni che impostano la gestione degli eventi del timer, caratteristica che consente di eseguire una subroutine ogni volta che sono trascorsi un numero determinato di secondi. La gestione degli eventi del timer può essere utilizzata per aggiornare ogni minuto l'ora visualizzata sullo schermo, oppure per salvare periodicamente un file su disco.

Le istruzioni fornite a questo scopo da QBasic sono ON TIMER GOSUB, TIMER ON, TIMER OFF e TIMER STOP. La gestione degli eventi del timer deve essere inizializzata con l'istruzione ON TIMER GOSUB e abilitata mediante TIMER ON, operazione che molto spesso si dimentica di eseguire.

Le istruzioni di seguito riportate inizializzano una routine di gestione degli eventi e consentono di eseguire ogni 60 secondi la subroutine in corrispondenza dell'etichetta MostraOrologio.

```
ON TIMER(60) GOSUB MostraOrologio    'inizializza gestione eventi
TIMER ON    'la attiva
```

Dato che l'utilizzo di questa caratteristica rallenta in modo considerevole il programma, è bene disattivarla durante l'esecuzione di operazioni critiche, quali le comunicazioni seriali di I/O, utilizzando l'istruzione TIMER OFF. Quando la gestione degli eventi del timer è disattivata, il programma viene eseguito nel modo consueto. Un'operazione analoga può essere eseguita con l'istruzione TIMER STOP; in questo caso il programma annota il numero di secondi passati, ma non esegue immediatamente la subroutine di gestione degli eventi. Tuttavia, quando viene eseguita un'altra istruzione TIMER ON, il programma esegue la subroutine.

Bibliografia

Norton, Peter, and Richard Wilton. *The New Peter Norton Programmer's Guide to the IBM PC & PS/2.* Redmond, Wash.: Microsoft Press, 1988.

Prata, Stephen, with Harry Henderson. *The Waite Group's Microsoft QuickBASIC Primer Plus.* Redmond, Wash.: Microsoft Press, 1990.

DATE$ (Istruzione e funzione)

Si consulti anche

TIME$ (Istruzione e funzione)

Scopo

L'istruzione DATE$ imposta la data di sistema, mentre la funzione DATE$ restituisce la data attuale. DATE$ può essere sfruttata per stampare una determinata data sulle intestazioni dei prospetti.

Sintassi

Istruzione:

```
DATE$ = espr
```

Funzione:

```
DATE$
```

espr è un'espressione stringa che deve avere uno dei seguenti formati: mm-gg-aa, mm-gg-aaaa, mm/gg/aa o mm/gg/aaaa. mm rappresenta il mese, gg il giorno, aa l'anno all'interno del ventesimo secolo e aaaa l'anno compreso il secolo. Non è necessario inserire uno zero prima del valore di un mese o di un giorno composto da una sola cifra.

Utilizzo

```
PRINT "Oggi è "; DATE$
```

Visualizza la data attuale.

```
DataOdierna$ = DATE$
```

Assegna la data attuale alla variabile stringa DataOdierna$.

```
DATE$ = "01-21-1992"
```

Imposta la data al 21 gennaio 1992.

```
DATE$ = DataOdierna$
```

Imposta la data in base a quella contenuta nella variabile DataOdierna$.

Descrizione

DATE$ può essere utilizzata sia come istruzione che come funzione; nel primo caso QBasic imposta la data, mentre nel secondo caso viene restituita la data attuale.

DATE$ come istruzione

L'istruzione DATE$ consente di impostare la data di sistema, utilizzata sia da QBasic che dal DOS. Questa istruzione richiede una sintassi diversa rispetto a quella della maggior parte delle istruzioni di QBasic. DATE$ deve essere utilizzata in un'istruzione di assegnazione, in cui il lato sinistro è la parola chiave DATE$ e il lato destro è una stringa che indica la nuova data.

DATE$ come funzione

La funzione DATE$ viene utilizzata per leggere la data attuale, che viene restituita nel formato mm-gg-aaaa, ovvero due cifre per il mese (da 1 a 12), un trattino, due cifre per il giorno (da 1 a 31), un trattino e quattro cifre per l'anno. Si noti che qualsiasi numero composto da una sola cifra comprende uno zero iniziale.

Errori

Se con l'istruzione DATE$ viene utilizzata una data non valida (ad esempio "Giugno 31" o "1979"), QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita". Se viene utilizzata un'espressione non stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Avvertenze

Non è possibile impostare una data precedente al 1 Gennaio 1980 o successiva al 31 Dicembre 2099.

Esempio

Il programma di seguito riportato utilizza l'istruzione DATE$ per modificare la data di sistema in modo che corrisponda a quella in cui cade il compleanno dell'utente e la funzione DATE$ per visualizzarla.

```
DEFINT A-Z

DIM mesi(12) AS STRING, numMesi AS INTEGER
DIM giorno AS INTEGER, anno AS INTEGER

FOR i% = 1 TO 12                'ottiene i nomi dei mesi dalle istruzioni DATA
    READ mesi(i%)
NEXT

INPUT "Quand'è il tuo compleanno <giorno-mese>"; compleanno$
vecchiaData$ = DATE$                     'memorizza la data attuale
DATE$ = RIGHT$(compleanno$, 2) + "-" + LEFT$(compleanno$, 2) + "-1992"
'cambia la data

numMesi = VAL(LEFT$(DATE$, 2))           'ottiene il mese da mm-gg-aaaa
giorno = VAL(MID$(DATE$, 4, 2))          'ottiene il giorno da mm-gg-aaaa
anno = VAL(RIGHT$(DATE$, 4))             'ottiene l'anno da mm-gg-aaaa
```

```
PRINT USING "Il tuo compleanno è il ## & ####"; giorno; mesi(numMesi); anno
DATE$ = vecchiaData$

DATA  "Gennaio","Febbraio","Marzo","Aprile","Maggio","Giugno","Luglio"
DATA  "Agosto","Settembre","Ottobre","Novembre","Dicembre"
```

ON TIMER GOSUB

Si consulti anche

TIMER ON/OFF/STOP

Scopo

L'istruzione ON TIMER GOSUB indica il sottoprogramma che viene eseguito ogni volta che è trascorso un numero determinato di secondi; può essere utilizzata per specificare la subroutine che visualizza e aggiorna l'ora a intervalli regolari.

Sintassi

```
ON TIMER(ora) GOSUB{riga | etichetta}
```

ora è un numero intero nell'intervallo da 1 a 86400 che indica il numero di secondi che devono trascorrere prima che il controllo passi a una subroutine.

riga o etichetta sono il numero di riga o l'etichetta che indicano la prima riga di una subroutine.

Utilizzo

```
ON TIMER(60) GOSUB AggiornaOrologio TIMER ON
```

Indica che la subroutine che inizia all'etichetta AggiornaOrologio viene eseguita ogni 60 secondi.

Descrizione

L'istruzione ON TIMER GOSUB specifica la posizione di una routine di gestione degli eventi del timer e il numero di secondi che devono trascorrere tra le esecuzioni della routine. Una routine di gestione degli eventi è una subroutine che viene eseguita quando si verifica un determinato evento. Questa istruzione indica informazioni dettagliate relative alla gestione degli eventi del timer, che esegue una subroutine ogniqualvolta è trascorso il numero di secondi indicati; la subroutine deve trovarsi nel programma principale.

Quando la subroutine di gestione degli eventi del timer inizia l'esecuzione, QBasic esegue implicitamente un'istruzione TIMER STOP, in modo che questa subroutine non venga eseguita in modo ricorsivo. Tuttavia, dato che QBasic utilizza TIMER STOP, se si verifica un altro evento del timer durante l'esecuzione della subroutine QBasic ne tiene traccia, eseguendo la subroutine in corrispondenza della successiva istruzione TIMER ON. Quando la subroutine individua un'istruzione RETURN, QBasic esegue l'istruzione TIMER ON, a meno che la subroutine non abbia eseguito un'istruzione TIMER OFF.

La subroutine di gestione degli eventi può anche terminare con un'istruzione RETURN che indichi un numero di riga o un'etichetta. L'istruzione fa in modo che l'esecuzione passi a una riga specificata del programma. Quest'ultima versione può provocare seri problemi, dato che l'intercettazione dell'evento può verificarsi in qualsiasi punto del programma; inoltre l'istruzione RETURN con un numero di riga o un'etichetta non può restituire il controllo a un sottoprogramma o a una funzione.

Errori

Se in un'istruzione ON TIMER GOSUB, viene indicato un numero di riga o un'etichetta che non esistono o che si trovano in una procedura, QBasic visualizza il messaggio di errore: “Etichetta non definita”. Se il numero di secondi specificati è superiore a 86400 (ovvero alle 24 ore) o inferiore a 1, QBasic visualizza il messaggio di errore: “Chiamata di funzione non lecita”. Se come numero di secondi viene specificata un'espressione non numerica, QBasic visualizza il messaggio di errore: “Mancata corrispondenza dei tipi”.

Suggerimenti

Dato che l'utilizzo della gestione degli eventi rallenta in modo considerevole il programma, è bene non sfruttare questa caratteristica nelle routine in cui la velocità è un fattore fondamentale.

Esempio

Il programma di seguito riportato utilizza l'istruzione ON TIMER GOSUB per specificare la subroutine che aggiorna la visualizzazione dell'ora ogni 60 secondi.

```
CLS
GOSUB ModificaOrologio                  'visualizza l'ora iniziale

ON TIMER(60) GOSUB ModificaOrologio    'specifica la posizione e il periodo
                                       'dell'intercettazione dell'evento
TIMER ON

LOCATE 5, 1: PRINT "Premi un tasto per terminare"
DO UNTIL INKEY$ <> ""           'abilita l'intercettazione dell'evento temporale
LOOP
END

ModificaOrologio:
   LOCATE 1, 1
   PRINT DATE$, LEFT$(TIME$, 5)         'visualizza ore e minuti
   RETURN
```

SLEEP

Si consulti anche

ON COM GOSUB, ON KEY(n) GOSUB, ON PEN GOSUB, ON PLAY GOSUB, ON STRING GOSUB, ON TIMER GOSUB, TIMER

Scopo

L'istruzione SLEEP sospende l'esecuzione del programma fino a quando è trascorso un numero determinato di secondi, viene premuto un tasto oppure si verifica un evento; può essere utilizzata ad esempio per sospendere un programma, in modo che l'utente abbia il tempo di leggere una schermata di output.

Sintassi

```
SLEEP [secondi]
```

secondi è un'espressione facoltativa di tipo intero che indica il numero di secondi per cui il programma viene sospeso.

Utilizzo

```
SLEEP
```

Sospende il programma fino a quando l'utente non preme un tasto o finché non si verifica un evento.

```
SLEEP 5
```

Sospende il programma per 5 secondi, oppure fino a quando l'utente non preme un tasto o finché non si verifica un evento.

Descrizione

L'istruzione SLEEP consente di inserire alcune pause in un programma, sospendendo l'esecuzione fino a quando non si verifica una delle seguenti situazioni:

- il numero specificato di secondi trascorre;
- l'utente preme un tasto;
- QBasic incontra un evento intercettabile.

Il numero di secondi indicato deve essere un'espressione di tipo intero; se è inferiore a 1 (o non viene specificato), il programma viene sospeso fino a quando non si verifica la seconda o la terza condizione indicata in precedenza. Se il numero è un valore in virgola mobile, SLEEP lo converte in un intero.

Questa istruzione risponde soltanto alla pressione dei tasti da parte dell'utente e agli eventi che si verificano dopo la sua esecuzione di SLEEP. Ne consegue che vengono ignorate tutte le sequenze di tasti presenti nel buffer della tastiera.

Gli eventi individuati possono interrompere l'istruzione SLEEP soltanto se sono stati attivati mediante l'istruzione ON corretta e non sono stati disattivati o sospesi da una corrispondente istruzione OFF o STOP.

Commenti

Dato che il numero indicato di secondi è un'espressione di tipo intero, la precisione è limitata ai secondi interi. Se è necessario effettuare una pausa per incrementi inferiori ai secondi, è sufficiente utilizzare la funzione TIMER in un'istruzione di ciclo.

Errori

Se in un'istruzione SLEEP viene specificata un'espressione non numerica, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Avvertenze

Dato che per interrompere un'istruzione SLEEP è sufficiente premere un tasto qualsiasi, compreso MAIUSC, CTRL, ALT, BLOC MAIUSC, BLOC NUM e BLOC SCORR, l'utente non può premere la combinazione di tasti MAIUSC+STAMP per produrre una copia della schermata durante una pausa creata da questa istruzione.

Esempio

Il programma di seguito riportato utilizza l'istruzione SLEEP per sospendere la visualizzazione, in modo che l'utente abbia il tempo di leggere la schermata.

```
CLS
PRINT "Hai"; FRE(-1); "byte di spazio libero per array. "
PRINT "Hai"; FRE(-2); "byte di spazio libero sullo stack. "

PRINT
PRINT "Ricopia questa informazione e premi un tasto per continuare. "
PRINT
PRINT "Questa schermata si autodistruggerà in 10 secondi ... "
SLEEP 10
CLS
```

TIME$ (Istruzione e funzione)

Si consulti anche

DATE$ (Istruzione e funzione), TIMER ON/OFF/STOP

Scopo

L'istruzione TIME$ imposta l'ora di sistema, mentre la funzione TIME$ restituisce l'ora attuale. TIME$ può essere sfruttata per stampare una determinata ora sulle intestazioni dei prospetti.

Sintassi

Istruzione:

```
TIME$ = espr
```

Funzione:

```
TIME$
```

espr è un'espressione stringa che deve avere uno dei seguenti formati:

hh	imposta l'ora; ai minuti e ai secondi viene assegnato il valore di default 0;
hh:mm	imposta l'ora e i minuti; ai secondi viene assegnato il valore di default 0;
hh:mm:ss	imposta l'ora, i minuti e i secondi.

hh rappresenta l'ora (da 0 a 23), mm i minuti (da 0 a 59) e ss i secondi (da 0 a 59). Si noti che non è necessario inserire gli zeri prima dei valori composti da una sola cifra.

Utilizzo

```
PRINT "Sono le "; TIME$
```

Visualizza l'ora attuale.

```
adesso$ = TIME$
```

Assegna l'ora attuale alla variabile stringa adesso$.

```
TIME$ = "10:35:30"
```

Imposta l'ora a 35 minuti e mezzo dopo le 10.

```
TIME$ = "10"
```

Imposta l'ora alle 10.

```
TIME$ = ora$
```

Imposta l'ora al valore contenuto nella variabile adesso$.

Descrizione

TIME$ può essere utilizzata sia come istruzione che come funzione; nel primo caso QBasic imposta l'ora, mentre nel secondo caso viene restituita l'ora attuale.

TIME$ come istruzione

L'istruzione TIME$ consente di impostare l'ora di sistema; l'orologio inizia a conteggiare partendo dall'ora specificata. Questa istruzione richiede una sintassi diversa rispetto a quella della maggior parte delle istruzioni di QBasic: deve essere utilizzata in un'istruzione di assegnazione, in cui il lato sinistro è la parola chiave TIME$ e il lato destro è una stringa che indica la nuova data.

TIME$ come funzione

La funzione TIME$ viene utilizzata per leggere l'ora attuale, che viene restituita nel formato hh-mm-ss, ovvero due cifre per l'ora (da 0 a 23), due punti, due cifre per i minuti (da 0 a 59), due punti e due cifre per i secondi (da 0 a 59). Si noti che qualsiasi numero composto da una sola cifra comprende uno zero iniziale.

Errori

Se con l'istruzione TIME$ viene utilizzata un'ora non valida (ad esempio "26:65:00"), QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita". Se viene utilizzata un'espressione non stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma qui di seguito riportato utilizza l'istruzione **TIME$** per impostare l'ora di sistema a 3:30 e la funzione **TIME$** per visualizzarla.

```
DEFINT A-Z

DIM ora AS INTEGER, minuti AS INTEGER, suffisso AS STRING

vecchiaOra$ = TIME$               'memorizza l'ora attuale del sistema
TIME$ = "3:30"                    'imposta l'ora alle 3:30 A.M.

ora = VAL(LEFT$(TIME$, 2))        'ottiene l'ora da oo:mm:ss
minuti = VAL(MID$(TIME$, 4, 2))   'ottiene i minuti da oo:mm:ss

IF ora = 0 THEN                   'mostra la mezzanotte come 12 A.M.
    ora = 12
    suffisso = "A.M."
ELSEIF ora < 12 THEN
    suffisso = "A.M."
ELSE
    ora = ora - 12
    suffisso = "P.M."
END IF

PRINT USING "L'ora attuale è ##:## &"; ora; minuti; suffisso

TIME$ = vecchiaOra$               'reimposta l'ora
END
```

TIMER

Si consulti anche

ON TIMER GOSUB, TIMER ON/OFF/STOP

Scopo

La funzione **TIMER**, che restituisce il numero di secondi trascorsi dalla mezzanotte, deve essere utilizzata per cronometrare la durata di una determinata operazione.

Sintassi

```
TIMER
```

Utilizzo

```
oraInizio$ = TIMER
```

Assegna il numero di secondi trascorsi dopo la mezzanotte alla variabile **oraInizio$**.

Descrizione

La funzione **TIMER** restituisce valori nell'intervallo da 0 a 86400. Talvolta è necessario conoscere il tempo necessario per eseguire una determinata operazione: questa funzione fornisce un metodo semplice per cronometrare le operazioni, dato che è sufficiente sottrarre il valore di **TIMER** all'inizio dell'operazione da quello ottenuto alla fine; il risultato ottenuto rappresenta la durata dell'operazione in secondi.

Commenti

Dato che il valore restituito da TIMER cambia sempre, spesso i programmatori utilizzano questa funzione come argomento nell'istruzione RANDOMIZE.

Avvertenze

Dato che la funzione TIMER restituisce il numero di secondi trascorsi dalla mezzanotte, a mezzanotte restituisce 0. Quindi, se un programma che utilizza TIMER viene eseguito a mezzanotte è necessario assicurarsi di modificare correttamente i risultati ottenuti.

Esempio

Il programma di seguito riportato utilizza la funzione TIMER per calcolare il tempo necessario a creare e visualizzare 2000 numeri casuali. La funzione TIMER viene anche sfruttata nelle istruzioni RANDOMIZE per inizializzare nuovamente il generatore di numeri casuali.

```
CLS
tempoInizio& = TIMER

RANDOMIZE TIMER
FOR i% = 1 TO 2000
    PRINT RND * 1000,
NEXT

tempoFine& = TIMER
PRINT
PRINT tempoFine& - tempoInizio&; "secondi trascorsi"
```

TIMER ON/OFF/STOP

Si consulti anche

ON TIMER GOSUB

Scopo

Le istruzioni TIMER attivano, disattivano o sospendono la gestione degli eventi del timer. L'istruzione TIMER ON può ad esempio essere utilizzata per attivare la gestione degli eventi che aggiorna un orologio visualizzato sullo schermo.

Sintassi

```
TIMER ON
TIMER OFF
TIMER STOP
```

Utilizzo

```
ON TIMER(60) GOSUB OgniMinuto
TIMER ON
```

Attiva la gestione degli eventi in modo che il programma esegua ogni 60 secondi la subroutine che inizia in corrispondenza dell'etichetta OgniMinuto.

```
TIMER OFF
```

Disattiva la gestione degli eventi.

Descrizione

Dopo aver richiamato l'istruzione ON TIMER GOSUB per indicare informazioni dettagliate relative alla gestione degli eventi del timer, è necessario attivare l'intercettazione degli eventi con l'istruzione TIMER ON. A questo punto, dopo ogni istruzione QBasic verifica se il tempo specificato nell'istruzione ON TIMER è effettivamente trascorso e in questo caso esegue la subroutine specificata nell'istruzione ON TIMER GOSUB.

L'istruzione TIMER OFF disattiva la gestione degli eventi del timer, fino a quando non viene eseguita un'altra istruzione TIMER ON.

L'istruzione TIMER STOP sospende la gestione degli eventi del timer, fino a quando non viene eseguita un'altra istruzione TIMER ON. Tuttavia, se si verifica un evento dopo questa istruzione, il programma imposta un flag in modo da "ricordarlo". Quando viene individuata un'altra istruzione TIMER ON, l'evento precedentemente intercettato viene gestito.

QBasic esegue un'istruzione TIMER STOP implicita ogniqualvolta viene inserita una routine di gestione degli eventi. Questo garantisce che ogni occorrenza dell'evento non provochi una gestione ricorsiva. Quando il programma incontra un'istruzione RETURN e ritorna al programma principale, imposta nuovamente la gestione degli eventi eseguendo un'istruzione TIMER ON.

Suggerimenti

Dato che le istruzioni di gestione degli eventi aumentano le dimensioni dei programmi eseguibili, rallentandoli considerevolmente, è meglio evitare di utilizzarle nelle routine in cui la velocità è un fattore fondamentale.

Esempio

Si consulti l'esempio riportato nella voce ON TIMER GOSUB, in questo capitolo.

Parte seconda

Multimedia

Capitolo 11

Grafica

INTRODUZIONE

Uno dei punti di forza di QBasic è rappresentato dalla serie di comandi di grafica. QBasic comprende inoltre il supporto per i colori e le modalità video ad alta risoluzione della nuova generazione di schede grafiche EGA e VGA.

Nella Tabella 11.1 sono riportate in ordine alfabetico le istruzioni e le funzioni grafiche di QBasic.

ADATTATORI VIDEO

Nella Tabella 11.2 sono elencati i tipi di adattatori video supportati da QBasic.

Gli adattatori MDPA e CGA erano quelli originariamente utilizzati nei PC IBM; il primo è in grado di visualizzare 25 righe per 80 colonne di testo su un video monocromatico, ma non dispone di alcuna capacità grafica. Il formato carattere 80 per 25 è la modalità testo di default per tutti gli adattatori video.

L'adattatore CGA supporta anche la grafica; quando viene utilizzato con un video a colori, è in grado di visualizzare 25 righe per 80 colonne di testo con 16 colori in primo piano e 8 di sfondo. Sono disponibili due modalità grafiche: la risoluzione media (320 per 200 pixel con 4 colori) e l'alta risoluzione (640 per 200 pixel con 2 colori).

L'adattatore HGC (Hercules Graphics Card) è in grado di visualizzare, oltre a 80 per 25 righe di testo, anche i grafici ad alta risoluzione (720 per 348 pixel) su video monocromatico. Per utilizzare questo adattatore in modalità grafica nell'ambiente QBasic, è necessario innanzitutto installare in memoria un driver residente (MSHERC.COM), che non è compreso nel pacchetto DOS fornito dalla Microsoft.

QBasic supporta inoltre gli adattatori Olivetti e AT&T, che sono in grado di visualizzare 80 per 25 righe di testo e 640 per 400 pixel di grafica. È possibile selezionare uno dei 16 colori di primo piano, ma il colore di sfondo è sempre nero.

Tabella 11.1 Le istruzioni e le funzioni grafiche di QBasic

Istruzione o funzione	Descrizione
CIRCLE	Traccia un cerchio, un'ellisse, un arco o un segmento in una posizione determinata dello schermo. È possibile specificare il raggio, il colore e il fattore di scala dell'oggetto.
COLOR	Imposta i colori di default di primo piano e dello sfondo per lo schermo in uso. Con alcune modalità video è anche possibile impostare il colore del bordo dello schermo.
DRAW	Esegue una stringa di istruzioni che definiscono le dimensioni, la forma, l'angolo e il colore di una figura.
GET (Grafica)	Copia un'immagina grafica dallo schermo in un array in memoria.
LINE	Traccia una linea che collega due serie di coordinate, oppure disegna un rettangolo con le due coordinate che indicano gli angoli opposti.
PAINT	Riempie un'area circoscritta con un colore o un motivo.
PALETTE, PALETTE USING	Assegna i valori dei colori agli attributi di visualizzazione nei sistemi EGA, VGA e MCGA.
PCOPY	Copia il contenuto di una pagina schermo in un'altra.
PMAP	Traduce tra le coordinate fisiche e quelle logiche.
POINT	Restituisce il colore del pixel specificato oppure le coordinate del cursore in modo grafico.
PRESET	Assegna nuovamente a un pixel sullo schermo il colore dello sfondo o quello specificato.
PSET	Assegna a un pixel sullo schermo il colore di primo piano o quello specificato.
PUT (Grafica)	Copia in un punto determinato dello schermo un'immagine da un array in memoria.
SCREEN (Funzione)	Restituisce il carattere o l'attributo di colore del carattere presente nella riga e nella colonna specificate.
SCREEN (Istruzione)	Imposta la modalità di schermo e la pagina schermo.
VIEW	Definisce le coordinate fisiche del viewport grafico attivo
WINDOW	Definisce il sistema di coordinate logiche in base al quale vengono tracciate le immagini.

I nuovi adattatori video hanno introdotto l'alta risoluzione e un numero superiore di colori, pur mantenendo la compatibilità con gli standard precedenti. L'EGA supporta la modalità testo MDPA e le modalità grafiche CGA, oltre ad aver aggiunto altre modalità grafiche. La risoluzione più alta possibile con un adattatore EGA è di 640 per 350 pixel con 16 colori.

I computer IBM della serie PS/2 sono stati introdotti sul mercato con due nuovi adattatori video; l'adattatore MCGA è simile al CGA, ma presenta altre due modalità grafiche: una a media risoluzione (320 per 200 pixel con 256 colori) e una ad alta risoluzione (640 per 480 pixel con 2 colori), non supporta però le modalità grafiche

Tabelle 11.2 Gli adattatori video supportati da QBasic

Nome	Abbreviazione
Monochrome Display and Printer Adapter	MDPA
Color Graphics Adapter	CGA
Hercules Graphics Card	HGC
Olivetti	-
AT&T	-
Enhanced Graphics Adapter	EGA
Multi-Color Graphics Array	MCGA
Video Graphics Array	VGA

EGA. L'adattatore VGA è in grado di visualizzare tutte le modalità degli adattatori MDPA, CGA, EGA e MCGA oltre alla propria modalità a 16 colori e 640 per 480 pixel. Altri colori per le schede MCGA e VGA vengono creati mediante l'utilizzo di monitor analogici, che mescolano i colori sullo schermo per produrre un'ampia gamma di sfumature.

PIXEL E RISOLUZIONE

Nel paragrafo precedente si è fatto riferimento alle capacità degli adattatori video nelle modalità grafiche, utilizzando i pixel per indicare il grado di risoluzione. Un *pixel* è il punto più piccolo che possa essere impostato o reimpostato singolarmente sullo schermo. Più pixel sono disponibili, superiore è la risoluzione delle immagini che possono essere disegnate. Ad esempio, un video a media risoluzione è in grado di visualizzare 320 per 200 pixel (64000); risoluzioni maggiori possono visualizzare 640 per 480 pixel (307200).

Un altro elemento che distingue gli adattatori è il colore, non tutti infatti possono visualizzarli; l'adattatore HGC, ad esempio, può visualizzare i pixel sullo schermo soltanto in due modi: acceso o spento. CGA ed EGA hanno risoluzioni che possono visualizzare i pixel in 2, 4 o 16 colori, mentre VGA e MCGA possono mescolare i colori, visualizzando in questo modo 256 colori diversi in modalità grafica a media risoluzione.

Nella Tabella 11.3 sono riportate le risoluzioni di visualizzazione, il corrispondente numero di colori disponibili e le modalità e gli adattatori richiesti per realizzarli.

MODALITÀ VIDEO

L'istruzione SCREEN consente di selezionare le modalità video disponibili in base all'adattatore utilizzato. Ad esempio, per passare alla modalità EGA di 640 per 350 pixel a 16 colori, è sufficiente eseguire l'istruzione SCREEN 9.

Tabella 11.3 La risoluzione grafica, i colori, la modalità e gli adattatori richiesti per riprodurli

Risoluzione	Colori	Modalità	Adattatori richiesti
320 per 200 pixel	4	1	CGA, MCGA, EGA, VGA
	16	7	EGA, VGA
	256	13	MCGA, VGA
640 per 200 pixel	2	2	CGA, MCGA, EGA, VGA
	16	18	EGA, VGA
640 per 350	4	10	EGA, VGA
	16	9	EGA, VGA
640 per 400	16	4	Olivetti, AT&T
720 per 348	2	3	HGC
640 per 480	2	11	MCGA, VGA
	16	12	VGA

La modalità video utilizzata definisce la risoluzione grafica e il numero di colori disponibili. QBasic offre 11 modalità per i grafici e una per il testo; nella colonna Modalità della Tabella 11.3 sono elencati i numeri da utilizzare con l'istruzione SCREEN per accedere alle diverse modalità grafiche dello schermo. Per specificare la modalità testo, è sufficiente utilizzare l'istruzione SCREEN 0.

COORDINATE DELLO SCHERMO

I pixel su uno schermo grafico vengono numerati in righe e colonnela; queste ultime vengono numerate da sinistra a destra, mentre le prime dall'alto verso il basso. Il numero di righe e di colonne disponibili dipende dalla risoluzione utilizzata: ad esempio la modalità video 2 ha 640 colonne per 200 righe.

Un singolo pixel può essere individuato utilizzando una coppia di coordinate che indicano la riga e la colonna; la prima, a cui si fa di solito riferimento come la coordinata orizzontale (*x*) è il numero di colonna, mentre la seconda, la coordinata verticale (*y*) è la riga. Nella Figura 11.1 viene illustrato l'intervallo di coordinate disponibile con la modalità video 9. Dato che i valori delle coordinate iniziano da 0, il pixel nell'angolo superiore sinistro dello schermo viene indicato dalle coordinate (0, 0), indipendentemente dalla modalità. Utilizzando la risoluzione dello schermo di 640 per 350 pixel (modalità video 9), il pixel posto al centro dello schermo si trova nel punto indicato dalle coordinate (319, 174).

DISEGNI

QBasic fornisce due istruzioni che consentono di disegnare singoli pixel: PSET e PRESET permettono di disegnare un pixel di un determinato colore; se non ne viene

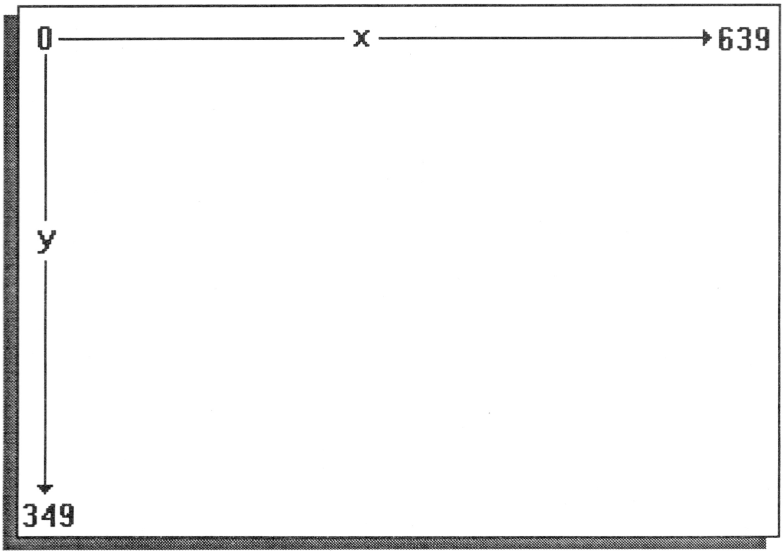

Figura 11.1 Le coordinate dei pixel nella modalità video 9

specificato uno, PSET utilizza il colore di primo piano in uso, mentre PRESET, che utilizza invece il colore di sfondo, può anche essere sfruttata per cancellare i pixel. La funzione POINT restituisce le informazioni relative a un pixel: se viene specificata una determinata posizione, restituisce il colore attuale del pixel corrispondente. Questa funzione può anche essere utilizzata per individuare la coordinata *x* o quella *y* in uso.

Altre istruzioni grafiche di QBasic utilizzano le coordinate per tracciare figure più elaborate. L'istruzione CIRCLE disegna un cerchio o un'ellisse, con centro in corrispondenza della coordinata specificata e con raggio uguale alla lunghezza indicata. È anche possibile indicare il colore del cerchio e tracciare archi e segmenti, invece di un cerchio completo. Per disegnare un cerchio che abbia centro in (50, 50), un raggio di 30 e un bordo verde, è sufficiente utilizzare l'istruzione:

```
CIRCLE (50, 50), 30, 2
```

L'istruzione LINE disegna una riga diritta tra due coppie di coordinate, o un rettangolo i cui vertici opposti sono posizionati in corrispondenza delle coordinate specificate.

Dopo aver tracciato la struttura di un oggetto, è possibile utilizzare l'istruzione PAINT per riempirlo con un determinato colore o motivo, fornendo le coordinate soltanto di un punto che si trova all'interno della forma da riempire. Per colorare in verde il cerchio disegnato in precedenza, è necessario utilizzare la seguente istruzione:

```
PAINT (50, 50), 2
```

Invece di utilizzare coordinate specifiche o assolute, è anche possibile utilizzare coordinate relative. Con un indirizzamento relativo, le coordinate *x* e *y* indicate in

un'istruzione non fanno riferimento alle coordinate della riga e della colonna in uso, ma alla distanza verticale e orizzontale del pixel corrente dalla posizione del cursore grafico. Quest'ultima è generalmente la posizione dell'ultimo punto disegnato dalle istruzioni grafiche precedenti. Tuttavia, se non è stata eseguita alcuna istruzione, la posizione del cursore è per default il centro dello schermo.

L'indirizzamento relativo risulta utile per la realizzazione di schermate elaborate, dato che modificando la posizione originale è possibile spostare una figura completa in una parte diversa dello schermo. Per utilizzare un indirizzamento relativo, è sufficiente aggiungere in un'istruzione grafica la parola chiave STEP prima delle coordinate.

COLORI

Tutte le istruzioni relative alla grafica accettano argomenti che consentono di indicare il colore del disegno in fase di realizzazione. Tuttavia, QBasic fornisce anche due istruzioni che consentono di controllare i colori di default della modalità video attivata.

VIEWPORT GRAFICO E FINESTRE

Sebbene sia possibile utilizzare tutto lo schermo per disegnare i grafici, è possibile limitare l'output soltanto a una parte più piccola, lasciando in questo modo la parte restante libera per l'output di testo; quest'area viene definita *viewport grafico*. Dopo che è stata utilizzata l'istruzione VIEW per specificare le coordinate che definiscono il viewport rettangolare, ogni istruzione grafica successiva deve fare riferimento alle posizioni all'interno del viewport, dato che qualsiasi altra coordinata all'esterno dell'area non viene disegnata.

Dopo aver definito un viewport, le coordinate *x* e *y* vengono misurate partendo dall'angolo superiore sinistro del viewport, non dello schermo. I programmi QBasic sono in grado di definire più di un viewport durante l'esecuzione, ma soltanto uno alla volta può essere attivo.

Le coordinate sull'asse verticale (*y*) vengono misurate partendo dalla parte superiore dello schermo; se si preferisce utilizzare il sistema di coordinate cartesiane, in cui le coordinate *y* vengono misurate partendo dalla parte inferiore dello schermo, e si desidera utilizzare le coordinate in un intervallo diverso da quello di default, è sufficiente utilizzare l'istruzione WINDOW, che consente di definire una serie di coordinate logiche che corrispondono alla scala della figura da tracciare. WINDOW mappa queste coordinate nel viewport fisico, per il quale le coordinate corrispondono alle posizioni effettive del pixel. Dopo aver chiamato WINDOW, le istruzioni grafiche quali LINE, CIRCLE e PAINT accettano argomenti dalla serie di coordinate logiche e QBasic li traduce nelle coordinate fisiche che corrispondono ai pixel sullo schermo.

Ad esempio, la finestra logica riportata nella Figura 11.2 può essere definita utilizzando la seguente istruzione:

```
WINDOW (0, 0) - (1000, 1000)
```

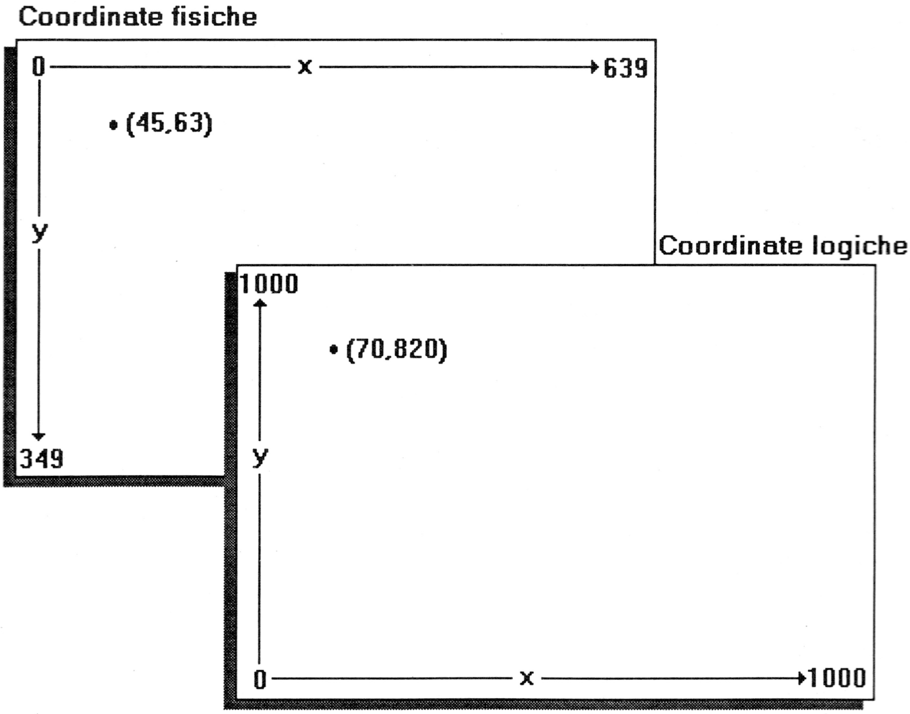

Figura 11.2 La conversione delle coordinate logiche in coordinate fisiche

per specificare un sistema di coordinate logiche che utilizza i valori *x* e *y* nell'intervallo da 0 a 1000 in un viewport di 640 per 350 pixel. Le coordinate verticali in questa finestra logica vengono tracciate partendo dalla parte inferiore dello schermo.

Per tracciare il punto (70, 820) nella finestra logica, è necessario utilizzare l'istruzione PSET, proprio come nel caso delle coordinate assolute, avvalendosi però delle coordinate logiche, ovvero:

```
PSET (70, 820)
```

QBasic traduce le coordinate logiche in coordinate fisiche, impostando in questo modo il pixel in corrispondenza della posizione fisica (45, 63).

Dato che la conversione delle coordinate logiche in fisiche viene eseguita automaticamente quando si utilizza WINDOW, raramente è necessario conoscere l'indirizzo effettivo di un pixel. Tuttavia è sempre possibile utilizzare la funzione PMAP, che restituisce la posizione fisica di qualsiasi coordinata logica.

L'istruzione WINDOW consente di realizzare velocemente disegni e grafici dai dati, ma prevede anche utilizzi avanzati. È possibile ad esempio definire una finestra logica che utilizza un intervallo di coordinate più piccolo rispetto a quelle che definiscono le dimensioni fisiche dello schermo; qualsiasi immagine grafica in seguito disegnata viene ingradita, quando confrontata con le stesse immagini tracciate utilizzando le

coordinate fisiche. Inoltre, allargando, contraendo o spostando le coordinate della finestre è possibile creare un effetto di "zoom" della visualizzazione.

ANIMAZIONE

Qualsiasi immagine tracciata sullo schermo può essere catturata e memorizzata dall'istruzione GET, che copia il contenuto di una determinata area dello schermo in un array in memoria. Per ripristinare l'immagine catturata in corrispondenza di una posizione diversa dello schermo, è sufficiente utilizzare l'istruzione PUT, che copia il contenuto di un array sullo schermo, partendo dalla posizione specificata. Copiando l'immagine sullo schermo diverse volte e in differenti posizioni, è possibile creare svariati effetti di animazione.

PAGINE VIDEO

Se l'adattatore dispone di memoria sufficiente, alcune modalità video consentono di gestire diverse schermate di informazioni, ognuna delle quali viene memorizzata in una *pagina video* separata. L'istruzione SCREEN consente di selezionare la pagina attiva, su cui operano le istruzioni grafiche, e la pagina visibile, che il programma visualizza; non è necessario che le due pagine coincidano. È possibile produrre un'animazione modificando le pagine visibili contenenti una stessa immagine che occupa una posizione leggermente diversa su ogni pagina.

QBasic fornisce inoltre l'istruzione PCOPY, che copia il contenuto di una pagina sorgente in quella di destinazione e può essere utilizzata per trasferire il contenuto di una pagina video in un'altra. Ad esempio, se abitualmente viene utilizzata una pagina particolare per l'animazione e si desidera salvare un particolare esempio di quella pagina, è possibile richiamare la procedura PCOPY per copiare l'immagine in una pagina che il programma non modifica.

VISUALIZZAZIONE DEI GRAFICI CON L'ISTRUZIONE DRAW

Le istruzioni grafiche finora illustrate utilizzano un sistema di grafica vettoriale, in cui gli estremi delle righe e delle curve vengono definiti in termini di coordinate dello schermo. L'istruzione DRAW fornisce un altro metodo per disegnare sullo schermo, permettendo di definire una riga in termini di direzione e lunghezza.

L'istruzione DRAW consente di controllare il movimento del cursore grafico, utilizzando diversi comandi per creare una figura che disponga delle dimensioni, della forma, del colore e dell'orientamento specificati. QBasic fornisce un'ampia gamma di comandi DRAW, in effetti si può dire che DRAW è di per sé un versatile linguaggio grafico.

Bibliografia

McGregor, Jim, and Alan Watt. *The Art of Graphics for the IBM PC.* Menlo Park, Calif.: Addison-Wesley, 1986.

Wilton, Richard. *Programmer's Guide to PC & PS/2 Video Systems.* Redmond, Wash.: Microsoft Press, 1987.

CIRCLE

Si consulti anche

COLOR, PAINT, SCREEN (Istruzione), WINDOW

Scopo

L'istruzione CIRCLE disegna un cerchio delle dimensioni specificate; può anche essere utilizzata anche per tracciare ellissi, archi e segmenti.

Sintassi

```
CIRCLE [STEP](x, y), raggio [, [colore] [, [inizio] [, [fine] [, aspetto]]]]
```

x e y sono le coordinate dello schermo relative al centro del cerchio. Se viene utilizzata la parola chiave STEP, queste coordinate sono relative alla posizione del cursore grafico; in caso contrario sono riferimenti assoluti.

raggio è la distanza dal centro del cerchio alla circonferenza ed è misurato nel sistema di coordinate specificato.

colore è l'attributo di colore assegnato al cerchio; se non ne viene specificato alcuno, viene utilizzato quello di primo piano in uso.

inizio e fine sono gli angoli (espressi in radianti) che indicano il punto iniziale e quello finale di un arco o di un segmento.

aspetto è il rapporto tra l'asse *y* e l'asse *x* che viene utilizzato per tracciare un'ellisse. Il valore di default corrisponde a quello che crea un cerchio perfetto in base alla modalità video attiva.

Utilizzo

```
CIRCLE (160, 100), 40
```

Traccia un cerchio con un raggio di 40 pixel nel centro dello schermo in modalità video 1. Dato che non viene specificato alcun colore, QBasic utilizza il colore di primo piano in uso.

```
CONST PI = 3.14159
CIRCLE STEP(50, -40), 60, 2, -PI, -PI * 2
```

Traccia un arco verde con il centro 50 pixel a destra e 40 pixel sopra la posizione del cursore grafico. I valori dei parametri inizio e fine sono negativi, quindi l'arco risultante viene congiunto al centro dai raggi, creando un semicerchio.

Descrizione

CIRCLE non traccia soltanto dei cerchi, ma può anche essere utilizzata per creare ellissi, archi e segmenti.

Per disegnare un cerchio, è necessario indicarne la posizione e le dimensioni; il primo valore viene definito utilizzando una coppia di coordinate per misurare la distanza orizzontale (*x*) e quella verticale (*y*) del centro del cerchio rispetto al vertice superiore sinistro dello schermo. Queste distanze vengono misurate in base al sistema di coordinate in uso, mentre l'intervallo dei valori possibili viene determinato dalla modalità grafica impostata da una precedente istruzione SCREEN. Ad esempio, per posizionare un cerchio al centro dello schermo, quando è attiva la modalità video 1 (grafici a media risoluzione), è sufficiente assegnare alle coordinate *x* e *y* rispettivamente i valori 160 e 100, mentre utilizzando le modalità video 11 e 12 VGA, è necessario utilizzare le coordinate 320 e 240.

Un altro metodo per posizionare un cerchio consiste nell'utilizzare la parola chiave STEP per specificare che le coordinate *x* e *y* indicano uno spostamento rispetto alla posizione del cursore grafico, che è la posizione dell'ultima figura tracciata da istruzioni grafiche. STEP fa in modo che CIRCLE esegua un indirizzamento relativo, operazione che presenta diverse utili applicazioni: un programma può riposizionare completamente una figura, semplicemente modificando le coordinate del punto iniziale.

Le dimensioni di un cerchio possono essere definite specificandone il raggio, ovvero la distanza dal centro alla circonferenza, e anche in questo caso la misura viene espressa nel sistema di coordinate specificato. I cerchi possono avere qualunque dimensione, tenendo conto però che qualsiasi parte collocata oltre i confini dello schermo non viene tracciata.

Aspetto

Per riprodurre l'aspetto di un cerchio, QBasic traccia in effetti un'ellisse o un ovale; ciò non sarebbe necessario se i pixel sullo schermo fossero uguali in altezza e larghezza, tuttavia, dato che i pixel sono più alti rispetto alla loro larghezza, un raggio di 50 pixel è più breve orizzontalmente che verticalmente. Quindi, un "vero" cerchio appare più alto che largo.

Dato che le dimensioni di un pixel sullo schermo dipendono dalla modalità video attiva, QBasic calcola un aspetto di default diverso in base alla modalità video in modo da poter creare cerchi "rotondi". Questa operazione viene eseguita utilizzando la formula di seguito riportata, dove xPixel e yPixel sono il numero massimo di pixel negli assi *x* e *y*.

```
aspetto = (yPixel / xPixel) * 4/3
```

Con una modalità video a media risoluzione (320 per 200 pixel), l'aspetto di default è 0,83333 (5/6); con una modalità ad alta risoluzione (640 per 200 pixel), è la metà di questo, ovvero 0,41667 (5/12). Ne consegue che, per tracciare un ellisse più lunga orizzontalmente che verticalmente, è necessario indicare un aspetto che sia inferiore rispetto a quello di default della modalità video attiva, mentre per tracciare un ellisse più alta che larga, è sufficiente indicare un valore di aspetto superiore a quello di default, come illustrato nella Figura 11.3. Se l'aspetto indicato è inferiore al valore di default, il raggio viene misurato lungo l'asse orizzontale; se invece l'aspetto è

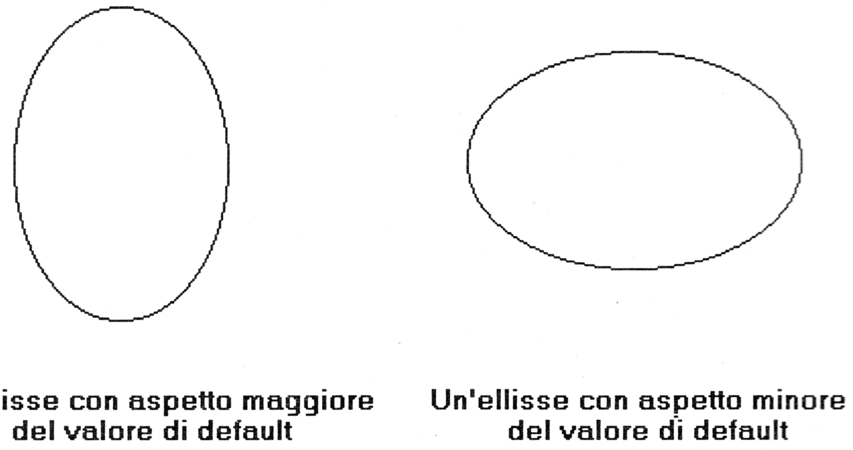

Figura 11.3 Gli effetti dell'indicazione dell'aspetto

superiore, il raggio viene misurato verticalmente. Quindi, il raggio specificato è sempre il valore dell'asse più lungo dell'ellisse.

Archi e segmenti

Per tracciare un arco, è sufficiente definire un punto iniziale e uno finale mediante angoli misurati in radianti, partendo dalla posizione delle ore tre sulla circonferenza del cerchio di cui l'arco è una parte. Il numero di radianti aumenta in senso antiorario, da 0 all'origine a 6,2832 (o 2* π), che è il numero di radianti che formano il cerchio completo, come illustrato nella Figura 11.4.

Se i valori di inizio e fine sono negativi, gli angoli che rappresentano vengono ancora considerati come positivi, ma l'istruzione CIRCLE traccia un raggio che collega il punto iniziale o quello finale sull'arco al centro del cerchio, creando in questo modo un segmento. Nella Figura 11.5 sono illustrati tre esempi dell'istruzione CIRCLE.

Suggerimenti

CIRCLE può sempre essere utilizzata con la parola chiave STEP, anche se in precedenza non è stata eseguita alcuna istruzione grafica. Quando QBasic modifica le modalità video, imposta il cursore grafico al centro della visualizzazione e traccia le successive coordinate relative partendo da quel punto.

Esempio

Una delle caratteristiche più utili di CIRCLE e della maggior parte delle istruzioni grafiche è la parola chiave STEP, che traccia le coordinate della figura in base alla posizione del cursore grafico. Il programma di seguito riportato utilizza STEP in un'istruzione CIRCLE per tracciare un'immagine di un extraterrestre. La posizione di questa complessa immagine può essere modificata semplicemente cambiando il valore delle coordinate iniziali.

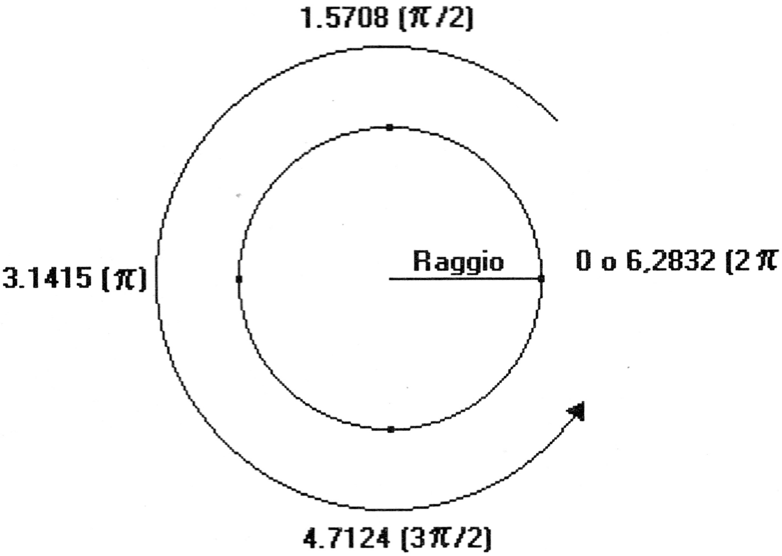

Figura 11.4 Le coordinate di inizio e fine dell'istruzione CIRCLE

```
DEFINT A-Z

x = 160: y = 120                         'coordinate iniziali
crl0 = 0: crl1 = 1: crl2 = 2: crl3 = 3   'valori per impostare i colori

SCREEN 1
LINE (0, 0)-(319, 199), crl3, BF         'colora l'intero schermo

'disegna la creatura extra-terrestre
CIRCLE (x, y), 35, crl0: PAINT STEP(0, 0), crl0, crl0
CIRCLE STEP(-30, -40), 20, crl0: PAINT STEP(0, 0), crl0, crl0
CIRCLE STEP(0, 0), 15, crl2, , , .5: PAINT STEP(0, 0), crl1, crl2
CIRCLE STEP(0, 0), 5, crl3, , , 6: PAINT STEP(0, 0), crl2, crl3
CIRCLE STEP(60, 0), 20, crl0: PAINT STEP(0, 0), crl0, crl0
CIRCLE STEP(0, 0), 20, crl0: CIRCLE STEP(0, 0), 15, crl2
PAINT STEP(0, 0), crl1, crl2: CIRCLE STEP(0, 0), 5, crl3, , , 6
PAINT STEP(0, 0), crl2, crl3
CIRCLE STEP(-15, 10), 11, crl0, 2.1, 3.3
CIRCLE STEP(0, 0), 6, crl0, 2, 3.3: PAINT STEP(-8, -4), crl0, crl0
CIRCLE STEP(-22, 4), 11, crl0, .01, 1.3
CIRCLE STEP(0, 0), 6, crl0, 6.1, 1.3: PAINT STEP(8, -4), crl0, crl0
CIRCLE STEP(x, y + 10), 30, crl2, , , .5: PAINT STEP(0, 0), crl1, crl2

FOR i! = .001 TO .5 STEP .07    'disegna i dettagli della bocca della creatura
    CIRCLE STEP(0, 0), 25, crl0, , , i!
NEXT i!
END
```

COLOR

Si consulti anche

PALETTE, SCREEN (Istruzione)

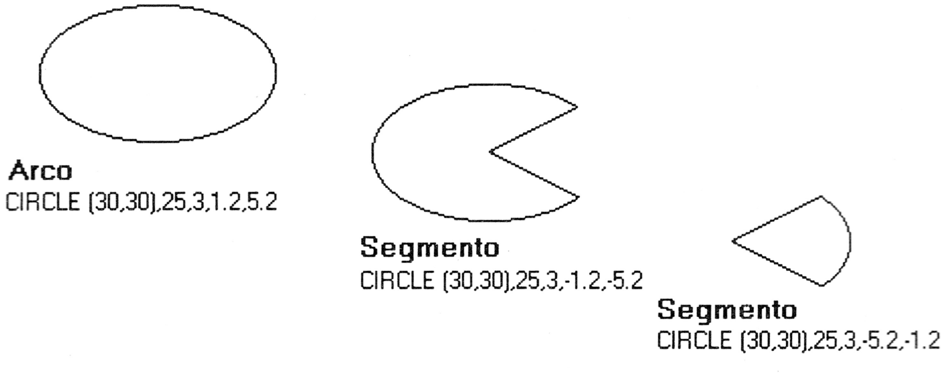

Figura 11.5 La creazione di archi e segmenti con l'istruzione CIRCLE

Scopo

L'istruzione COLOR seleziona gli attributi di visualizzazione di un oggetto sullo schermo. Se il computer dispone di un video a colori, questi attributi vengono tradotti nei colori effettivi, mentre con i video monocromatici controllano l'intensità degli oggetti visualizzati. COLOR può anche essere utilizzata per impostare gli attributi di visualizzazione quando è attiva la modalità testo.

Sintassi

```
COLOR [primo piano] [,[sfondo] [,bordo%]]   (Modalità video 0)
COLOR [sfondo] [,palette]                   (Modalità video 1)
COLOR [primo piano] [,sfondo]               (Modalità video 7-10)
COLOR [primo piano]                         (Modalità video 4, 12, 13)
```

primo piano è un valore intero che imposta il colore del testo e dei grafici visualizzati sullo schermo.

sfondo è un valore intero che imposta il colore di sfondo dello schermo.

bordo è un valore intero che imposta il colore del bordo dello schermo.

palette è un numero intero utilizzato nella modalità video 1 che specifica quale dei due insiemi di attributi di colore deve essere utilizzato.

Utilizzo

```
SCREEN 8
COLOR 1, 14
```

Assegna il blu al colore di primo piano e il giallo chiaro a quello di sfondo.

Descrizione

Il numero e lo scopo degli argomenti forniti a COLOR variano in base alla modalità video selezionata; tutti gli argomenti che indicano un attributo specificato da un colore sono riportati nella Tabella 11.4.

Tabella 11.4 I numeri degli attributi di COLOR

Attributo	Colore
0	Nero
1	Blu
2	Verde
3	Ciano
4	Rosso
5	Magenta
6	Marrone
7	Bianco
8	Grigio scuro
9	Blu chiaro
10	Verde chiaro
11	Ciano chiaro
12	Rosso chiaro
13	Magenta chiaro
14	Giallo
15	Bianco brillante

Si noti che i colori corrispondenti agli attributi da 8 a 15 sono identici a quelli corrispondenti da 0 a 7, tranne per il fatto che un'intensità maggiore li fa apparire più luminosi.

L'istruzione COLOR esegue compiti diversi, che vengono determinati dalla modalità video attiva.

COLOR con la modalità video 0

La modalità video 0 è quella di testo. Quando è attiva, COLOR controlla l'aspetto dei caratteri stampati e il colore dello sfondo. I numeri per il colore di primo piano possono essere compresi nell'intervallo da 0 a 31. I numeri da 0 a 15 sono quelli di attributo per i colori, che sono riportati nella Tabella 11.4; i numeri da 16 a 31 sono gli stessi colori nello stesso ordine, ma lampeggianti. I numeri per il colore di sfondo sono compresi nell'intervallo da 0 a 7 e rappresentano i primi otto colori, riportati nella Tabella 11.4; non è possibile né specificarli con un'intensità più alta, né indicare quelli che lampeggiano.

Immettere un'istruzione COLOR in modalità testo non ha alcun effetto sui caratteri già visualizzati, tuttavia, l'output successivo ha i colori di primo piano e di sfondo indicati. L'istruzione CLS deve essere utilizzata per cancellare lo schermo, in modo da attribuire il nuovo colore di sfondo.

Ovviamente i colori non vengono visualizzati, se si dispone di un video monocromatico. Il testo visualizzato con i colori di primo piano nell'intervallo da 8 a 15 appaiono più luminosi rispetto a quelli nell'intervallo da 0 a 7. Con gli adattatori MDPA e HGC, il colore di primo piano 1 visualizza il testo sottolineato, mentre il colore di sfondo 9 visualizza un testo sottolineato luminoso.

I colori di sfondo da 0 a 6 non hanno alcun effetto con un video monocromatico, ma il numero 7 utilizzato con il colore di sfondo 0 inverte il colore di primo piano con quello di sfondo (normalmente bianco su nero).

I computer che utilizzano l'adattore CGA (Color Graphics Adapter) possono anche utilizzare l'istruzione COLOR per impostare il colore del bordo dello schermo, attribuendogli qualsiasi valore nell'intervallo da 0 a 15.

COLOR con la modalità video 1

La modalità video 1 è la modalità grafica a media risoluzione (320 per 200 pixel). Si noti che, quando è attiva questa modalità, COLOR non può impostare il colore di primo piano, anche se è in grado di attribuire allo sfondo dello schermo un colore della gamma completa riportata nella Tabella 11.4. Richiamando questa istruzione, il colore della visualizzazione cambia immediatamente.

Le istruzioni grafiche, quali LINE, CIRCLE e PAINT, utilizzate nella modalità video1, possono tracciare un disegno utilizzando uno dei quattro attributi di colore (da 0 a 3). Il colore assegnato a questi attributi dipende dalla palette specificata nell'istruzione COLOR; sono disponibili due palette, illustrate nella Tabella 11.5.

Tabella 11.5 I numeri degli attributi di COLOR per ogni palette

Attributo	Palette 0	Palette 1
0	Colore di sfondo attivo	Colore di sfondo attivo
1	Verde	Ciano
2	Rosso	Magenta
3	Marrone	Bianco

Se alle istruzioni grafiche non viene assegnato un argomento colore esplicito, QBasic utilizza quello corrispondente all'attributo 3; ad esempio il rettangolo:

```
LINE (50, 35) - (216, 60)
```

viene tracciato in marrone se è attiva la palette 0 e in bianco se invece è attiva la palette 1. Quando la modalità video 1 viene selezionata per la prima volta, l'impostazione di default di COLOR è sfondo nero e palette 1 (ciano, magenta e bianco)

COLOR con le modalità video da 7 a 10

Nelle modalità video da 7 a 10, i colori di primo piano e di sfondo sono, per default, bianco brillante su nero. Le modalità video 7, 8 e 9 consentono di effettuare una selezione tra 16 attributi (numeri da 0 a 15), che indicano i colori del primo piano e dello sfondo.

Queste modalità possono essere utilizzate soltanto con adattotori EGA o VGA e un video a colori che consenta di visualizzare contemporaneamente 16 colori. Nelle

modalità 7 e 8, questi colori vengono selezionati dai 16 attributi di colore elencati nella Tabella 11.4, nella modalità 9 è possibile effettuare la selezione da una palette estesa di 64 attributi di colore (numerati da 0 a 63), a patto che l'adattatore utilizzato abbia più di 64 KB di memoria e che il video sia in grado di visualizzare 64 colori; la modalità video 10, la modalità EGA e VGA monocromatica, supporta quattro attributi relativi a pseudocolori di primo piano, riportati nella Tabella 11.6, e nove, riportati nella Tabella 11.7, per i pseudocolori di sfondo.

Tabella 11.6 Gli attributi disponibili per il primo piano, quando è attiva la modalità video 10

Attributo	Descrizione del pseudocolore
0	Nessun pixel
1	Intensità normale
2	Lampeggiante
3	Alta intensità

Tabella 11.7 Gli attributi disponibili per lo sfondo, quando è attiva la modalità video 10

Numero di colore	Descrizione del pseudocolore
0	Nessun pixel
1	Lampeggio tra spento e intensità normale
2	Lampeggio tra spento e alta intensità
3	Lampeggio tra spento e intensità normale
4	Intensità normale, nessun lampeggio
5	Lampeggio tra intensità normale e alta intensità
6	Lampeggio tra alta intensità e spento
7	Lampeggio tra alta intensità e intensità normale
8	Alta intensità, nessun lampeggio

COLOR con le modalità video 4, 12 e 13

Nelle modalità video 4, 12 e 13 è possibile selezionare soltanto gli attributi di primo piano. Per modificare il colore di default dello sfondo (nero) nelle modalità 12 e 13, è sufficiente utilizzare l'istruzione PALETTE per assegnare il colore prescelto all'attributo 0. Le modalità 4 e 12 supportano 16 attributi di primo piano, mentre la modalità 13 ne supporta 256. Ogni attributo può essere assegnato a uno dei 262144 colori disponibili utilizzando l'istruzione PALETTE o PALETTE USING.

Errori

L'istruzione COLOR non è valida nella modalità video 2, 3 e 11. Se viene utilizzata in questi casi, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita". Questo messaggio viene visualizzato anche quando i parametri impostati non rientrano nell'intervallo specificato dalla modalità video attiva.

Esempio

L'istruzione COLOR varia in base all'adattatore utilizzato dal sistema e alla modalità definita dal programma con l'istruzione SCREEN. Se viene creato un programma che deve essere compatibile tra i vari sistemi, è necessario prendere in considerazione i diversi tipi di adattatore. Il programma di seguito riportato contiene l'istruzione COLOR in tre formati diversi, quello utilizzato dipende dai valori immessi dall'utente.

```
ON ERROR GOTO ModoErrato

PRINT "Qaule tipo di adattatore video stai utilizzando ?"
INPUT "<Inserire 1 per CGA, 2 per EGA e 3 per VGA o MCGA>:", adatt%

MostraColori:
SELECT CASE adatt%
CASE 1          'mostra i colori di fondo del modo 1 per CGA
    SCREEN 1
    PRINT "Puoi usare uno dei 15 colori di fondo."
    PRINT "Premi un tasto per vederli."
    DO WHILE INKEY$ = "": LOOP
    CLS
    FOR i% = 0 TO 14
        COLOR i%
        LOCATE 10, 10: PRINT "Colore:"; i%
        SLEEP 1
    NEXT i%
CASE 2          'mostra i colori di fondo e di primo piano per EGA
    SCREEN 7
    PRINT "Puoi usare uno dei 15 colori di fondo e"
    PRINT "uno dei 15 colori di primo piano."
    PRINT "Premi un tasto per vederli."
    DO WHILE INKEY$ = "": LOOP
    CLS
    FOR i% = 0 TO 14
        COLOR , i%
        CLS
        FOR j% = 0 TO 14
            COLOR j%
            PRINT "Primo piano:"; j%; "Fondo:"; i%
        NEXT j%
        SLEEP 1
    NEXT i%
CASE 3          'mostra i colori di primo piano per VGA e MCGA
    SCREEN 13
    PRINT "Puoi usare uno dei 256 colori di"
    PRINT "primo piano."
    PRINT "Premi un tasto per vederli."
    DO WHILE INKEY$ = "": LOOP
    CLS
    FOR i% = 0 TO 255
        COLOR i%
        PRINT "Colore:"; i%
    NEXT i%
```

```
END SELECT
END

ModoErrato:
IF ERR = 5 THEN
     PRINT "Il tuo sistema non supporta modalità video per quell'adattatore."
     INPUT "Inserire un numero diverso: ", adatt%
     RESUME MostraColori
ELSE
     ON ERROR GOTO 0
END IF
```

DRAW

Si consulti anche

VARPTR$, WINDOW

Scopo

L'istruzione DRAW traccia la figura specificata in un'espressione stringa; utilizza un linguaggio macro che consente di descrivere le linee in termini relativi, specificando la distanza e la direzione del movimento del cursore grafico.

Sintassi

```
DRAW espressionestringa
```

espressionestringa è un letterale di tipo stringa, una variabile stringa o un'espressione stringa che contiene una sequenza dei comandi necessari per disegnare un oggetto. Le stringhe letterali devono essere racchiuse tra doppi apici. I comandi possono essere immessi indifferentemente in lettere maiuscole o minuscole e separati con spazi o punti e virgola.

Utilizzo

```
DRAW "C4 U45 R75 D45 L75"
```

Traccia un rettangolo alto 45 unità e largo 75 nel colore 4 (rosso, per default).

```
DRAW riquadro$
```

Traccia la figura indicata dai comandi presenti nella variabile stringa riquadro$.

```
DRAW "C=" + VARPTR$(colore%)
```

Imposta il colore per le successive istruzioni DRAW in base a quello specificato dal valore della variabile di tipo intero colore%.

Descrizione

Il comando DRAW accetta una stringa che contiene un elenco di comandi macro, che indicano il movimento del cursore, la scala, la rotazione e il colore. Di seguito vengono riportati i comandi macro raggruppati per funzione.

Movimento del cursore

Il comando Mx,y è l'unico comando di DRAW che consente di specificare le coordinate, spostando il cursore in corrispondenza della posizione indicata dalle coordinate *x* e *y*. Se il valore di *x* è preceduto da un segno più (+) o meno (-), il movimento del cursore è relativo alla posizione attuale del cursore grafico; in caso contrario, QBasic presuppone che *x* e *y* siano le coordinate assolute di una posizione sullo schermo. In entrambi i casi, QBasic traccia una linea compresa tra la posizione del cursore e quella indicata, che diventa la nuova posizione del cursore grafico. Ad esempio, l'istruzione:

```
DRAW "M10,25"
```

traccia una linea partendo dalla posizione del cursore fino al punto (10, 25) sullo schermo. Invece l'istruzione:

```
DRAW "M-10,-25"
```

traccia una linea dalla posizione del cursore fino a un punto che si trova 10 pixel a sinistra e 25 pixel sopra la posizione attuale.

Il comando B indica che il successivo comando di movimento sposta il cursore grafico senza però che quest'ultimo tracci una linea. Per default, tutti i movimenti del cursore hanno come risultato una linea tracciata dal punto precedente alla nuova posizione, ma se un comando viene preceduto da B, produce una mossa "vuota", in cui la posizione attuale del cursore viene aggiornata, ma non viene tracciata alcuna linea tra quest'ultima e la posizione precedente.

Il comando N indica che il comando di movimento che segue non aggiorna la posizione del cursore grafico; l'istruzione DRAW traccia la linea, ma poi il cursore torna alla posizione di partenza.

Nella Tabella 11.8 sono riportati i comandi che consentono di spostare il cursore in diverse direzioni. È possibile controllare di quanto si sposta il cursore nella direzione prescelta, specificando la lunghezza della linea risultante in *unità*. Per default, un'unità è uguale a un pixel; questo valore può tuttavia essere modificato, cambiando il fattore di scala in uso con il comando S. Se non viene indicata alcuna lunghezza, DRAW sposta il cursore di una unità nella direzione specificata.

Fattore di scala

Sn, il comando relativo alla scala, indica un numero (nell'intervallo da 1 a 255) che viene diviso per quattro per determinare il numero di pixel di spostamento per ogni unità della distanza fornita in un comando di movimento. Il valore di scala di default è 4, che rende un'unità di movimento uguale a un singolo pixel.

Il comando S risulta utile quando si desidera tracciare diverse copie della stessa figura con dimensioni diverse.

Rotazione

DRAW consente di ruotare gli assi verticali e orizzontali di una figura fino a 360 gradi. Per default, il comando U sposta il cursore grafico verso l'alto a un angolo di 0 gradi; tuttavia, se gli assi vengono ruotati di 45 gradi, il comando U traccia una linea

Tabella 11.8 I comandi che consentono di spostare il cursore in modo grafico

Comando	Descrizione
Un	Movimento verso l'alto di n unità
Dn	Movimento verso il basso di n unità
Rn	Movimento verso destra di n unità
Ln	Movimento verso sinistra di n unità
En	Movimento in diagonale verso l'alto di n unità e verso destra di n unità
Fn	Movimento in diagonale verso il basso di n unità e verso destra di n unità
Gn	Movimento in diagonale verso il basso di n unità e verso sinistra di n unità
Hn	Movimento in diagonale verso l'alto di n unità e verso sinistra di n unità

diagonale verso l'alto e verso sinistra. Gli altri comandi di direzione vengono influenzati in modo analogo. L'istruzione DRAW comprende due comandi di rotazione, che influenzano soltanto i comandi successivi e non ruotano nessuna linea che sia già stata tracciata.

Il comando An ruota gli assi in passi di 90 gradi. Il valore di n può essere un numero tra 0 e 3: DRAW moltiplica il valore per 90, in modo da calcolare il numero di gradi della rotazione. Gli angoli vengono misurati in senso antiorario, per cui le ore 12:00 corrispondono a 0 gradi.

Il comando TAN ruota gli assi di n gradi, dove n è un numero nell'intervallo da -360 a 360. Il valori positivi di n ruotano gli assi in senso antiorario, mentre i valori negativi in senso orario. Gli spostamenti degli assi rimangono validi anche per i comandi successivi.

Colore

DRAW comprende un comando che consente di specificare il colore della linea tracciata. L'intervallo possibile dei valori è determinato dalla modalità video attiva; se non viene indicato alcun colore, l'istruzione DRAW assume per default il colore di sfondo in uso.

Il comando Cn imposta il colore delle linee tracciate dai comandi di direzione e di movimento, assegnando il colore associato al valore n.

Il comando Pc,b esegue la stessa operazione dell'istruzione PAINT, ovvero riempie l'interno di una figura con il colore specificato da c, mentre b indica il colore del bordo.

Se è stata appena tracciata una figura da colorare, è sufficiente spostare il cursore grafico fuori dal bordo e all'interno della figura; in caso contrario non può essere eseguita la funzione di riempimento. Il metodo migliore per effettuare questa operazione consiste nell'eseguire una mossa vuota, come nel seguente esempio:

```
DRAW "C2 R50 D30 L50 U30"      'disegna un rettangolo verde
DRAW "BF10" 'sposta il cursore all'interno della figura
DRAW "P3,2" 'la riempie con il colore ciano
```

Sottostringhe

Il comando X consente di specificare le sottostringhe che DRAW deve eseguire dalla stringa di comando in uso. Questa è una caratteristica molto potente del linguaggio macro di DRAW. Per sfruttare questo comando, è necessario utilizzare la funzione VARPTR$ per fornire a DRAW l'indirizzo della sottostringa da eseguire (VARPTR$ restituisce sotto formato di stringa un puntatore alla variabile). Ad esempio, quando DRAW incontra l'istruzione:

```
"X" + VARPTR$(nave$)
```

in una stringa più ampia, esegue i comandi specificati nella variabile stringa nave$, prima di completare la stringa originale. La variabile nave$ può contenere altri comandi X, eseguendo quindi altre sottostringhe, che a loro volta possono contenere comandi X e così via.

Variabili

In un'istruzione DRAW, le variabili numeriche possono essere utilizzate in due modi: il primo consiste nell'utilizzo della funzione STR$ per convertire i valori in stringhe, come riportato nelle seguenti istruzioni:

```
mossa$ = "BM" + STR$(x%) + "," + STR$(y%)
DRAW mossa$
```

È anche possibile utilizzare la funzione VARPTR$ per fornire come argomento una variabile numerica all'interno dell'istruzione DRAW. In questo caso, il comando deve avere un segno di uguale (=) tra esso e l'espressione VARPTR$. Ad esempio l'istruzione:

```
DRAW "G=" + VARPTR$(pixel%)
```

traccia una linea diagonale verso il basso e a sinistra. La lunghezza della linea (in unità) viene specificata dal valore della variabile di tipo intero pixel%.

Esempio

Il comando DRAW risulta utile quando è necessario tracciare forme utilizzando frattali oppure figure costruite da versioni più piccole della stessa figura. Il programma di seguito riportato consente di sperimentare questa tecnica; modificando i parametri che regolano la lunghezza della linea e il numero di lati nel sottoprogramma Fiocco, è possibile tracciare diversi motivi intricati e affascinanti.

```
DECLARE SUB Frattale (n AS INTEGER)
DECLARE SUB Fiocco (n AS INTEGER, lati AS INTEGER)
DECLARE SUB IncAng (incremento AS INTEGER)

DIM SHARED angolo AS INTEGER
SCREEN 9
Fiocco 120, 9
END

SUB Fiocco (n AS INTEGER, lati AS INTEGER) STATIC
    muove$ = "BM+" + STR$(n \ 2) + ",+" + STR$(n \ 2)
    DRAW muove$                                'muove il cursore grafico
    ruota% = 360 \ lati
```

```
    FOR i% = 1 TO lati
        Frattale n                      'disegna una porzione del frattale
        IncAng ruota%
        DRAW "TA=" + VARPTR$(angolo)      'ruota gli assi
    NEXT i%
END SUB

SUB Frattale (n AS INTEGER)
    IF n < 4 THEN
        DRAW "U=" + VARPTR$(n)  'disegna una porzione del frattale
    ELSE
        Frattale n / 3        'chiamata ricorsiva del sottoprogramma Frattale
        IncAng -60
        DRAW "TA=" + VARPTR$(angolo)
        Frattale n / 3
        IncAng 120
        DRAW "TA=" + VARPTR$(angolo)
        Frattale n / 3
        IncAng -60
        DRAW "TA=" + VARPTR$(angolo)
        Frattale n / 3
    END IF
END SUB


SUB IncAng (incremento AS INTEGER) STATIC
    angolo = angolo + incremento
    IF angolo > 360 THEN
       angolo = angolo - 360
    ELSEIF angolo < 0 THEN
       angolo = angolo + 360
    END IF
END SUB
```

GET (Grafica)

Si consulti anche

PUT (Grafica), SCREEN (Istruzione), VIEW, WINDOW

Scopo

L'istruzione GET copia un'immagine grafica dallo schermo in un array. La corrispondente istruzione PUT copia il contenuto dell'array nuovamente sullo schermo, nella posizione indicata. Queste istruzioni consentono di spostare un'immagine da una parte dello schermo in un'altra, senza doverla disegnare nuovamente.

Sintassi

```
GET [STEP](x1,y1)-[STEP](x2,y2), array[(indice)]
```

x1, y1, x2 e y2 sono le coordinate dello schermo che indicano i vertici opposti dell'area rettangolare da copiare. La parola chiave STEP specifica che la posizione a cui le coordinate fanno riferimento è relativa alla posizione del cursore grafico.

array è il nome dell'array numerico in cui è memorizzata l'immagine. È possibile specificare l'elemento di array in cui inizia la memorizzazione dell'immagine, inserendo il valore di indice tra parentesi; se questo valore viene omesso, GET memorizza l'immagine a partire dal primo elemento dell'array.

Utilizzo

```
GET STEP(20, -5) -STEP(xInc, yInc), immagine&(0, 5)
```

Definisce un rettangolo il cui vertice superiore sinistro si trova 20 pixel verso destra e 5 pixel sopra rispetto al cursore grafico. Il vertice inferiore destro del rettangolo dista dalla nuova posizione del cursore di xInc pixel orizzontali e yInc pixel verticali. Il contenuto di questo rettangolo viene quindi copiato nella colonna 5 dell'array a due dimensioni immagine&.

Descrizione

Questa istruzione non deve essere confusa con la versione di I/O su file di GET, che viene utilizzata per leggere i dati da una file ad accesso casuale. L'istruzione GET relativa ai grafici legge i dati dallo schermo e li copia in un array in memoria. Generalmente questi dati sono sotto forma di una figura tracciata da una o più istruzioni grafiche.

Le coordinate del rettangolo sorgente da cui viene copiata l'immagine vengono specificate nello stesso modo dell'istruzione LINE. È possibile utilizzare il numero di pixel effettivi misurati dal vertice superiore sinistro del viewport in uso. Se è stata eseguita un'istruzione WINDOW, è possibile utilizzare il sistema di coordinate logiche da essa definite.

Dato che GET copia ogni pixel dal rettangolo sorgente nell'array di destinazione, è necessario assicurarsi che quest'ultimo sia sufficientemente grande. Per calcolare le dimensioni dell'array richiesto, è possibile utilizzare la seguente formula:

```
num_byte = 4 +INT(((larghezza) * bit_per_pixel/bit_piani) + 7) /8 * bit_piani
* altezza
```

larghezza è l'estensione orizzontale del rettangolo sorgente; per calcolarla in base alle coordinate *x*, è sufficiente utilizzare la seguente formula:

```
larghezza = (x2 -x1) + 1
```

In modo analogo, è possibile ottenere il valore di altezza, ovvero l'estensione verticale del rettangolo sorgente, utilizzando la seguente formula:

```
altezza = (y2 -y1) + 1
```

bit_per_pixel si basa sul numero di colori disponibili sullo schermo.

bit_piani è il numero di diverse aree in memoria utilizzate per memorizzare il colore di un pixel. Nella Tabella 11.9 sono riportati i valori di entrambi questi parametri per ogni modalità video.

Nelle modalità video 7, 8, 9 e 12, l'adattatore divide i quattro bit per il colore di ogni pixel tra quattro righe separate di bit, definite *piani di bit*; la modalità video 10 divide i due bit per il colore di ogni pixel tra due piani di bit. I valori elencati nella Tabella 11.9 tengono conto di questa divisione.

La formula riportata in precedenza restituisce le dimensioni di un array in byte. Il numero effettivo di elementi richiesti nell'array dipende dal tipo di array utilizzato. Nella Tabella 11.10 sono elencate le dimensioni degli elementi (in byte) per i differenti tipi di array.

Tabella 11.9 I bit per pixel e i piani di bit per ogni modalità video

Modalità video	Bit per pixel	Piani di bit
1	2	1
2	1	1
3	1	1
4	1	4
7	4	4
8	4	4
9	2	2 (se l'adattatore EGA ha 64 KB)
	4	4 (se l'adattatore EGA ha più di 64 KB)
10	2	2
11	1	1
12	4	4
13	8	1

Tabella 11.10 Le dimensioni degli elementi per i tipi diversi array numerici

Tipo di array	Byte per elemento
Intero	2
Intero lungo	4
A precisione semplice	4
A precisione doppia	8

Per copiare il rettangolo sorgente definito dalle coordinate (50, 20) e (69, 34) ad esempio in modalità 9, occorre dimensionare un array di tipo intero composto da 92 elementi, come determinato dal seguente calcolo:

```
(4 + INT(((20) * (4 / 4) + 7) /8) * 4 * 15) /2 = 92
```

Dopo aver effettuato questo calcolo, è possibile eseguire le istruzioni di seguito riportate per copiare il rettangolo sorgente in array.

```
DIM array(0 TO 91) AS INTEGER
GET (50, 20) - (69, 34), array
```

Esempio

Si consulti la voce PUT per un esempio che utilizza sia GET che PUT.

LINE

Si consulti anche

PALETTE, SCREEN (Istruzione)

Scopo

L'istruzione LINE utilizza una linea diritta di un determinato colore e stile per congiungere due punti sullo schermo. Le coordinate specificate possono anche contrassegnare i vertici di un rettangolo, che può essere riempito con un colore.

Sintassi

```
LINE [[STEP](x1,y1)]-[STEP](x2,y2) [,[colore] [,[B[F]] [,stile]]]
```

x1, y1, x2 e y2 sono le coordinate dello schermo che indicano l'inizio e la fine della linea da tracciare. La parola chiave STEP specifica che le posizioni a cui le coordinate fanno riferimento sono relative alla posizione del cursore grafico.

colore è un attributo di colore che imposta il colore della riga.

La lettera B indica a LINE di disegnare un rettangolo utilizzando le coordinate specificate per il vertice superiore sinistro e quello inferiore destro. Se viene aggiunta una F, viene invece creato un rettangolo riempito del colore selezionato.

stile è un valore a 16 bit che indica lo stile della linea tracciata.

È necessario definire i punti iniziali e finali nell'istruzione LINE utilizzando le coordinate assolute o quelle relative; tutti gli altri argomenti sono facoltativi. Se viene tralasciato un argomento che non è l'ultimo nell'elenco dei parametri, è necessario inserire una virgola per contrassegnarne la posizione.

Utilizzo

```
LINE (120, 100) - (30, 15)
```

Disegna una linea (utilizzando il colore di primo piano in uso) dalla posizione (30, 15) a (120, 100).

```
LINE -(100, 50), , B
```

Disegna, in base al colore di primo piano in uso, un rettangolo, che ha un vertice in corrispondenza della posizione del cursore e quello opposto in corrispondenza della coordinata assoluta (100, 50).

```
LINE(x%, y%) - (x% + 20, y% + 12), c%, BF
```

Disegna un riquadro alto 12 pixel e largo 20, in cui il vertice superiore sinistro viene specificato dalle variabili di tipo intero x% e y%. Il riquadro viene riempito in base al colore indicato dal valore contenuto in c%.

Descrizione

L'intervallo valido delle coordinate per le posizioni specificate nell'istruzione LINE dipende dalle dimensioni della modalità video attiva. Se viene utilizzato il parametro B o quello BF, QBasic traccia un rettangolo i cui vertici opposti diagonalmente sono i punti specificati, in caso contrario LINE traccia una linea diritta tra i due punti. Se entrambe le coordinate fanno riferimento a un punto che si trova all'esterno del viewport definito, QBasic non visualizza un messaggio di errore, ma si limita a tracciare quella parte della linea che si trova all'interno del viewport.

Non è fondamentale l'ordine delle due posizioni specificate. In effetti, le due istruzioni di seguito riportate disegnano la stessa linea.

```
LINE (35, 18) - (178, 86)
LINE (178, 86) - (35, 18)
```

Anche il numero di colori disponibili dipende dalla modalità video attiva. I numeri relativi ai colori corrispondono a quelli degli attributi utilizzati dall'istruzione COLOR. Tuttavia, se il computer dispone di un adattatore EGA, VGA o MCGA, è possibile rimappare i numeri degli attributi ad altri colori, utilizzando l'istruzione PALETTE.

Per default, le linee e le strutture dei rettangoli sono piene e ininterrotte. Tuttavia, è anche possibile tracciare linee intermittenti o punteggiate in una ampia gamma di stili. È possibile specificare un argomento di tipo intero i cui bit vengono letti (da sinistra verso destra) come una maschera che determina se un particolare pixel debba essere o meno illuminato; generalmente l'argomento viene specificato come numero esadecimale. Ad esempio, &H5555 (21845 decimale) crea una linea punteggiata con una serie alternata di pixel, mentre &H0707 produce una riga intermittente, come riportato nella Figura 11.6.

È possibile definire un motivo sia per le linee singole che per i lati del rettangolo, esso non influenza il modo in cui LINE riempie i riquadri; persino se le linee che compongono i lati del rettangolo sono spezzate, il colore non esce dai confini del riquadro.

Esempio

Una potente caratteristica dell'istruzione LINE è la parola chiave STEP, che consente di tracciare figure complesse, senza dover calcolare le coordinate assolute di ogni punto a cui fanno riferimento. Il programma di seguito riportato utilizza LINE per tracciare un grafico a barre tridimensionale. Il fatto di poter tracciare le coordinate di ogni barra con STEP rende il programma più semplice, dato che ogni barra è identica nella forma e si differenzia soltanto nell'altezza.

```
SCREEN 9
LINE (0, 0)-(639, 349), 9, BF        'colora l'intero schermo di blu
VIEW (32, 4)-(607, 345), 0, 15       'definisce un viewport
WINDOW SCREEN (0, 0)-(255, 191)      'definisce un sistema di coordinate logiche

READ numArticoli%
DIM articolo!(numArticoli%)

xInt% = 240 \ numArticoli%                'inizializza l'intervallo per x
xLargh% = xInt% - 10
zLargh% = xLargh% * 2

FOR i% = 1 TO numArticoli%
    READ articolo!(i%)
    IF articolo!(i%) > max! THEN max! = articolo!(i%)
NEXT i%

yInt! = 160 / max!: i% = 1: cl% = 1

FOR x% = 15 TO 239 STEP xInt%             'disegna le barre del grafico
    yLung% = articolo!(i%) * yInt!
    LINE (x%, 180)-(x% + xLargh%, 180 - yLung%), cl%, BF
    LINE -STEP(5, -3), cl%: LINE -STEP(-xLargh%, -0), cl%
```

Esadecimale	Binario	Tipo di linea
&H5555	0101010101010101	■ ■ ■ ■ ■ ■ ■ ■
&H0707	0000011100000111	▬ ▬ ▬ ▬

Figura 11.6 Alcuni esempi di stili di linea

```
    LINE -STEP(-5, 3), cl%: LINE -STEP(xLargh%, 0), cl%
    LINE -STEP(5, -3), cl%: LINE -STEP(0, yLung%), cl%
    LINE -STEP(-5, 3), cl%: i% = i% + 1: cl% = cl% + 1
    IF cl% = 7 THEN cl% = 9
NEXT x%
r$ = INPUT$(1)
END

DATA 12
DATA 356.21,216.53,99.9,25.8,176,282.75
DATA 303.4,111.11,168.3,271.9,333.3,88
```

PAINT

Si consulti anche

COLOR, PALETTE, SCREEN (Istruzione)

Scopo

L'istruzione PAINT riempie l'area di una figura chiusa con un determinato colore. É possibile specificare un colore pieno oppure uno retinato.

Sintassi

```
PAINT [STEP] (x,y)[,[colore] [,[colorebordo] [,sfondo]]]
```

x e y sono le coordinate dello schermo dove inizia il disegno. La parola chiave facoltativa STEP specifica che le coordinate sono relative alla posizione del cursore grafico.

colore può essere un'espressione numerica o un'espressione di tipo stringa: la prima indica il numero di attributo corrispondente al colore che riempie l'area, la seconda definisce un nuovo motivo di riempimento. Se quest'ultimo parametro viene omesso, PAINT riempie l'area con il colore di primo piano attivo.

colorebordo è il numero corrispondente al colore del bordo dell'area da riempire. Se questo parametro viene tralasciato, PAINT assegna per default lo stesso colore del parametro colore.

sfondo è un'espressione di tipo stringa composta da un solo carattere che definisce un motivo di sfondo utilizzabile per riempire un'area che era stata già riempita in precedenza.

Utilizzo

```
PAINT (319, 174), 14, 4
```

Sposta il cursore al centro dello schermo (quando è attiva la modalità 9) e riempie l'area con il colore giallo (attributo 14), fino a quando non individua un bordo rosso o finché non è riempito tutto il viewport.

Descrizione

Per richiamare PAINT allo scopo di colorare una figura, è necessario innanzitutto specificare le coordinate di una posizione all'interno della figura in corrispondenza della quale deve iniziare l'operazione di riempimento. L'intervallo dei valori possibili viene determinato dalla modalità grafica attiva; ad esempio, è possibile utilizzare le coordinate (159, 99) per posizionare il cursore al centro dello schermo quando è attiva la modalità 1 (grafici a media risoluzione); per quanto riguarda invece le modalità 11 e 12, il centro dello schermo deve essere indicato con le coordinate (319, 239).

Se viene utilizzata l'istruzione WINDOW per definire un proprio sistema di coordinate logiche, QBasic traduce queste coordinate inviate a PAINT in posizioni fisiche dei pixel e le traccia correttamente.

PAINT presuppone che il bordo della figura sia già stato tracciato e utilizza un particolare algoritmo che assegna a tutti i pixel all'interno del bordo l'attributo di colore specificato. Se non viene indicato un colore di riempimento, PAINT utilizza quello di primo piano; se invece non viene specificato il colore del bordo, PAINT riempie lo spazio definito da un bordo con lo stesso colore utilizzato per il riempimento. È necessario assicurarsi che l'area da riempire sia totalmente chiusa, in caso contrario il colore "si spande".

PAINT utilizza lo stack del programma per registrare la posizione dei pixel già impostati, quindi se i lati della figura sono particolarmente frastagliati, il programma potrebbe visualizzare il messaggio di errore: "Spazio stack esaurito". In questo caso, è sufficiente utilizzare l'istruzione CLEAR per aumentare le dimensioni dello stack.

Creazione di motivi

Per indicare il colore pieno utilizzato da PAINT per il riempimento, è sufficiente specificare un numero; i valori sono limitati all'intervallo di colori disponibili nella modalità video attiva. Utilizzando tuttavia un valore stringa, è possibile specificare una combinazione di colori per produrre un'ampia gamma di ombreggiature, retini e motivi. Questo procedimento è noto come *creazione di motivi.*

Per comprenderne il funzionamento, è necessario innanzitutto sapere in che modo QBasic crea i colori pieni nella modalità video in uso. Nella modalità grafica a 4 colori (modalità video 1), ogni pixel è rappresentato da 2 bit di memoria dello schermo che indicano uno dei quattro colori. Nella modalità grafica a 2 colori (modalità video 2), ogni pixel è rappresentato da 1 bit. Nella modalità grafica a 16 colori (modalità video 7, 8, 9 e 12), ogni pixel è rappresentato da 4 bit, ognuno dei quali definisce un attributo di colore; i 4 bit specificano una combinazione di blu, rosso, verde e l'intensità del colore. L'impostazione di ogni bit viene combinata con le impostazioni di un altro bit per determinare il colore del pixel, ad esempio un pixel impostato con i valori di bit di 1010 viene visualizzato nel colore 10, ovvero un verde chiaro. Si noti che il valore

del numero binario 1010 è 10 in decimale. Come illustrato nella Tabella 11.11, i numeri dei 16 colori disponibili indicano l'impostazione dei bit che crea il colore.

Se si desidera creare un proprio motivo per PAINT, è sufficiente creare una *casella motivo*, ovvero una griglia rettangolare di pixel che definisce il motivo. La griglia deve sempre essere lunga 8 bit e alta fino a 64 bit. Quando PAINT viene utilizzata con la casella motivo, quest'ultima viene ripetutamente tracciata fino a riempire l'area.

Per la modalità video 1, ogni riga della griglia deve contenere la rappresentazione di quattro pixel per riempire otto bit. Per la modalità video 2, ogni riga deve contenere la rappresentazione di 8 pixel. Nelle modalità video 7, 8, 9 e 12, i 4 bit che rappresentano ogni pixel vengono memorizzati in quattro aree diverse della memoria, definite piani di bit. Ogni byte della memoria video (o riga in una griglia) contiene quindi un bit da 8 diversi pixel.

Tabella 11.11 Le impostazioni dei bit per i colori

Impostazione bit	Valore decimale	Colore
0000	0	Nero
0001	1	Blue
0010	2	Verde
0011	3	Ciano
0100	4	Rosso
0101	5	Magenta
0110	6	Marrone
0111	7	Bianco
1000	8	Grigio
1001	9	Blu chiaro
1010	10	Verde chiaro
1011	11	Ciano chiaro
1100	12	Rosso chiaro
1101	13	Magenta chiaro
1110	14	Giallo
1111	15	Bianco brillante

Ad esempio, per creare un'ombreggiatura intermedia di verde è possibile progettare una casella motivo in cui i pixel adiacenti sono di due diverse intensità di verde, come riportato nella Figura 11.7.

Per specificare una casella motivo nell'istruzione PAINT, è sufficiente fornire una stringa in cui ogni byte contiene i bit di ogni piano di bit. Ad esempio, per creare la casella motivo definita dalla griglia nella Figura 11.7 è necessario indicare una stringa di 4 caratteri in cui il valore di ogni carattere è la somma dei bit in uno dei piani di bit:

```
casella$ = CHR$(0) + CHR$(255) + CHR$(0) + CHR$(85)
```

Il programma riportato nel seguito colora tre cerchi, utilizzando questo motivo per riempirne uno con una sfumatura di verde che si trova tra le due intensità fornite da QBasic.

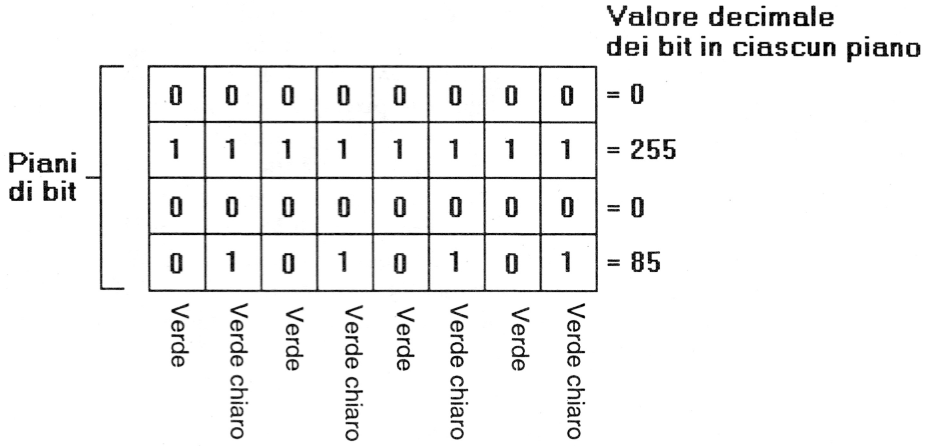

Figura 11.7 Un motivo di una casella relativo a un ombreggiatura intermedia di verde per PAINT

Ovviamente, è possibile mescolare colori diversi nella casella motivo, non soltanto differenti sfumature dello stesso colore; questo consente di creare molte e interessanti sfumature e retini.

PAINT consente inoltre di creare le stringhe di motivi con una lunghezza massima di 64 caratteri, in cui ogni serie di 4 caratteri definisce il motivo per le linee successive di 8 pixel. Anche se la realizzazione di caselle motivo può talvolta risultare un compito noioso, si ricordi che è possibile utilizzare gli stessi motivi in molti programmi diversi.

```
titolo$ = CHR$(0) + CHR$(255) + CHR$(0) + CHR$(85)

SCREEN 9
FOR x% = 100 TO 500 STEP 200
    CIRCLE (x%, 174), 80, 4
NEXT x%

PAINT (100, 174), 2, 4
PAINT (300, 174), titolo$, 4
PAINT (500, 174), 10, 4
END
```

Esempio

La creazione di motivi consente di ottenere molti più colori e sfumature, rispetto a quelle generalmente disponibili in base alla modalità video attiva. Questo procedimento consente di creare effetti più interessanti e realistici che con i colori pieni. Il programma di seguito riportato traccia un paesaggio di montagna e aggiunge quel tocco di colore e di retino necessario per creare l'atmosfera.

Uno dei problemi riscontrati con la creazione dei motivi riguarda l'armonizzazione del colore della struttura con il retino interno in fase di riempimento. Questo programma risolve il problema ritracciando il bordo (con l'istruzione LINE) con

un'opzione di stile che corrisponde al motivo di bit della casella. Tutto ciò può produrre notevoli effetti, soprattutto se la figura ha una struttura complessa.

```
SCREEN 8
DIM titolo$(1 TO 8)                        'stile della linea per il quarto motivo
linStile% = &H55AA
PAINT (319, 99), 15
VIEW (14, 6)-(624, 123), 15, 0
FOR i% = 1 TO 21
    READ x1%, y1%, x2%, y2%, cl%            'legge le coordinate della linea
    LINE (x1%, y1%)-(x2%, y2%), 0           'disegna il profilo del paesaggio
NEXT i%

FOR i% = 1 TO 8                             'legge il numero di byte nel titolo
    READ j%
    FOR k% = 1 TO j%
        READ byte%                          'legge i dati del pixel
        titolo$(i%) = titolo$(i%) + CHR$(byte%) 'costruisce il titolo colorato
    NEXT k%
NEXT i%


FOR i% = 1 TO 8
    READ x%, y%                             'legge le coordinate per PAINT
    PAINT (x%, y%), titolo$(i%), 0          'colora l'area racchiusa
NEXT i%

RESTORE                                     'reimposta il puntatore ai dati
FOR i% = 1 TO 21
    READ x1%, y1%, x2%, y2%, cl% 'legge le coordinate della linea e il colore
    LINE (x1%, y1%)-(x2%, y2%), cl%         'cancella il profilo
NEXT i%

LINE (400, 90)-(430, 87), 4, , linStile% 'ritocca il bordo del quarto motivo
LINE -(520, 84), 4, , linStile%
LINE -(610, 82), 4, , linStile%
END
'coordinate dello schermo e colori delle linee
DATA 0,75,130,66,5,130,66,150,65,9,150,65,215,60
DATA 9,215,60,305,65,9,305,65,380,55,9,380,55,395
DATA 50,9,395,50,415,55,9,415,55,460,62,9,460,62
DATA 520,68,9,520,68,610,75,9,0,58,130,66,9,305
DATA 65,610,80,5,615,98,305,98,1,305,98,380,93,1
DATA 380,93,400,90,1,400,90,430,87,2,430,87,520,84
DATA 2,520,84,610,82,2,0,90,400,90,1,0,98,610,112
DATA 2,615,104,285,104,14

'valori per i motivi dei titoli
DATA 4,255,0,0,255,4,255,0,0,0
DATA 8,0,255,170,0,0,255,85,0
DATA 8,0,170,85,0,0,85,170,0
DATA 8,85,170,255,170,170,85,255,85
DATA 4,255,0,255,0,4,255,0,255,170
DATA 4,0,255,255,255

'coordinate delle aree da colorare
DATA 10,10,10,95,305,113,600,95,305,85
DATA 380,60,10,63,605,110
```

PALETTE, PALETTE USING

Si consulti anche

COLOR, SCREEN (Istruzione)

Scopo

L'istruzione PALETTE modifica un colore nella palette disponibile; PALETTE USING cambia invece tutti i colori della palette. Questa seconda istruzione può essere utilizzata per produrre effetti grafici sorprendenti.

Sintassi

```
PALETTE [attributo, colore]
PALETTE USING array(indice)
```

attributo è il numero di attributo del colore da modificare; **colore** è il numero del nuovo colore a cui l'attributo viene assegnato.

array è il nome dell'array contenente i valori dei colori da assegnare alla palette in uso; **indice** identifica l'elemento dell'array in corrispondenza del quale inizia la procedura di riassegnazione.

Utilizzo

```
PALETTE 6, 14
```

Modifica il colore associato all'attributo 6 da marrone in giallo.

```
PALETTE USING clr&(9)
```

Modifica qualsiasi attributo, assegnando il colore contenuto in un elemento dell'array clr&. Questa operazione inizia in corrispondenza dell'elemento il cui indice è 9, quindi l'attributo 0 viene assegnato al colore contenuto in clr&(9).

Descrizione

L'insieme di colori disponibili in una determinata modalità video viene definito *palette*. In base alla modalità attiva, una palette può contenere 2, 4, 16 o persino 256 colori, a ognuno dei quali viene assegnato un proprio numero di attributo, che indica la posizione del colore nella palette. Nei computer che dispongono degli adattatori EGA, VGA o MCGA, è tuttavia possibile riassegnare questi numeri di attributo ad altri colori in modo che, ad esempio, una figura originariamente tracciata in rosso possa essere modificata in blu.

Quando viene selezionata per la prima volta una modalità video, QBasic assegna un numero di colore di default a ogni attributo presente nella palette relativa. Nelle modalità video con 16 colori (ovvero 7, 8, 9 e 12), la palette di default viene configurata come quella riportata nella Tabella 11.12.

PALETTE consente di modificare il numero di colore assegnato a qualsiasi singolo attributo nella palette. L'esecuzione di questa operazione dà come risultato che ogni pixel sullo schermo con quell'attributo viene visualizzato nel nuovo colore. Questa modifica si rispecchia anche nell'output sullo schermo dei grafici successivi; qualsiasi istruzione grafica in cui viene specificato un attributo (ad esempio LINE, CIRCLE o PAINT) viene tracciata utilizzando il colore assegnato all'attributo.

Tutte le modifiche apportate rimangono valide fino a quando non si cambia la modalità video o non si cancellano le modifiche mediante un'altra istruzione PALETTE. Se questa istruzione viene utilizzata senza argomenti, vengono nuovamente impostati i valori di default, in base alla modalità video attiva.

Tabella 11.12 I numeri degli attributi e dei colori nelle modalità schermo a 16 colori

Attributo	Numero del colore	Colore
0	0	Nero
1	1	Blue
2	2	Verde
3	3	Ciano
4	4	Rosso
5	5	Magenta
6	6	Marrone
7	7	Bianco
8	8	Grigio
9	9	Blu chiaro
10	10	Verde chiaro
11	11	Ciano chiaro
12	12	Rosso chiaro
13	13	Magenta chiaro
14	14	Giallo
15	15	Bianco brillante

PALETTE USING consente di utilizzare una singola istruzione per modificare i colori assegnati a tutti gli attributi nella palette. Per sfruttare questa istruzione, è necessario innanzitutto dimensionare un array contenente un numero di elementi almeno pari a quello dei colori presenti nella palette; tali elementi devono essere del tipo corretto per contenere i valori di colore. Nelle modalità EGA è sufficiente un array di tipo intero, le modalità VGA e MCGA richiedono invece interi lunghi a causa del metodo che utilizzano per creare i colori. Per specificare i valori dei colori della nuova palette, è sufficiente assegnarli agli elementi dell'array. Se non si desidera modificare un attributo, è necessario assegnare all'elemento il valore -1, invece del numero del colore.

Sebbene gli esempi precedenti utilizzino soltanto i numeri di colore nell'intervallo da 0 a 15, è possibile assegnare 64 colori diversi nelle modalità EGA e 262144 in quelle VGA e MCGA.

I numeri dei colori per le modalità EGA variano da 0 a 63. Sebbene i diversi colori disponibili nelle modalità VGA e MCGA siano 262144, i relativi numeri non variano da 0 a 262143, ma è necessario calcolare il valore di ogni colore; in effetti ogni colore in queste modalità è una determinata miscela di intensità dei colori blu, verde e rosso. L'intensità del colore può variare da 0 a 63: per calcolare il valore di un colore, è sufficiente utilizzare la formula di seguito riportata, in cui blu, verde e rosso sono i valori che indicano l'intensità.

```
valore = (65536 * blu) + (256 * verde) + rosso
```

Se ad esempio si desidera creare un blu puro brillante, è necessario utilizzare nel calcolo un'intensità di 63 per il valore blu e un'intensità di 0 per verde e rosso. Il risultato di questo calcolo è 4128768.

Esempio

Dato che l'istruzione PALETTE USING modifica i colori dello schermo, senza tracciare nuovamente la figura visualizzata, viene spesso utilizzata per produrre effetti di animazione. Il programma di seguito riportato esegue cicli di successioni regolari di modifiche dei colori per creare l'illusione di un movimento attraverso un tunnel.

```
SCREEN 9
DIM array(0 TO 32) AS INTEGER
PALETTE USING array(0)                  'imposta ogni colore al colore di fondo
FOR i% = 1 TO 32: array(i%) = (i% MOD 15) + 1: NEXT     'riempie l'array per
                                                        'la palette
LINE (0, 0)-(639, 399), 1, BF
cl% = 0                     'cambia il colore di primo piano in quello di fondo
FOR i% = 440 TO 10 STEP -10          'colora lo schermo con cerchi concentrici
    cl% = cl% + 1
    IF cl% > 15 THEN cl% = 1    'attribuisce un colore diverso ad ogni cerchio
    IF cl% = 8 THEN cl% = 9
    CIRCLE (320, 200), i%, 0
    PAINT (320, 200), cl%, 0
NEXT i%
PALETTE                                    're-imposta i colori di default
DO
    FOR i% = 15 TO 1 STEP -1
        a1% = array(i%): array(i%) = 0     'imposta il primo colore della
                                           'nuova palette a 0
        PALETTE USING array(i%)            'cambia la palette
        array(i%) = a1%
        IF INKEY$ = CHR$(27) THEN EXIT FOR
    NEXT i%
LOOP UNTIL i%
END
```

PCOPY

Si consulti anche

SCREEN (Istruzione)

Scopo

L'istruzione PCOPY copia una pagina video in un'altra. I computer che dispongono di determinati adattatori video possono passare tra otto diverse pagine schermo.

Sintassi

```
PCOPY sorgente, destinazione
```

sorgente è un intero che indica il numero della pagina video da copiare; destinazione è un intero che specifica il numero della pagina video in cui viene copiata quella sorgente.

Utilizzo

```
PCOPY 3, 0
```

Copia il contenuto della pagina 3 nella pagina 0, ovvero la prima pagina video.

Descrizione

Per utilizzare PCOPY, il computer deve disporre di un adattatore che supporti diverse pagine video; EGA e VGA possono visualizzare fino a otto pagine video nelle modalità 0, 7, 8, 9 e 10; CGA e MCGA possono visualizzare lo stesso numero di pagine nella modalità video0. I computer che dispongono di un adattatore monocromatico possono visualizzare soltanto una pagina video e non supportano l'istruzione PCOPY.

Errori

Se l'istruzione PCOPY viene utilizzata in una modalità che non supporta diverse pagine video, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita". Tale messaggio che viene visualizzato anche quando si cerca di sfruttare PCOPY per copiare in o da un numero di pagina che si trova oltre l'intervallo disponibile in base alla modalità video attiva.

Suggerimenti

Dato che PCOPY funziona soltanto in modalità testo 0 e nelle modalità grafiche 7 o superiori, non è possibile utilizzare in modalità grafica questa istruzione sui computer che dispongono degli adattatori CGA, Hercules, Olivetti o AT&T.

Esempio

Quando viene utilizzata con l'istruzione SCREEN, PCOPY è in grado di memorizzare i grafici e il testo su una pagina video, che può quindi essere copiata nel punto desiderato della visualizzazione attiva. Il programma di seguito riportato comprende una schermo di aiuto normalmente non visualizzato, che l'utente può però visualizzare premendo un tasto. Per ripristinare lo schermo precedente, è sufficiente premere un altro tasto.

```
'questo programma non funzionerà con adattatori monocromatici
SCREEN 0, , 1, 0         'imposta coma pagina attiva la pagina 1, come pagina
                         'visualizzata la pagina 0

'disegna lo schermo di Aiuto sulla pagina attiva
COLOR , 1: CLS : COLOR , 0
FOR riga% = 11 TO 16
    LOCATE riga%, 29: PRINT SPACE$(24):
NEXT riga%
COLOR 0, 3

'disegna un riquadro
LOCATE 10, 28: PRINT CHR$(201); STRING$(22, 205); CHR$(187);
FOR riga% = 11 TO 14
    LOCATE riga%, 28: PRINT CHR$(186); SPACE$(22); CHR$(186);
NEXT riga%
LOCATE riga%, 28: PRINT CHR$(200); STRING$(22, 205);
PRINT CHR$(188): : LOCATE 12, 33

'visualizza il messaggio
FOR i% = 1 TO 14: READ a%: PRINT CHR$(a%); : NEXT
LOCATE 13, 29
FOR i% = 1 TO 21: READ a%: PRINT CHR$(a%); : NEXT
LOCATE 14, 35
FOR i% = 1 TO 9: READ a%: PRINT CHR$(a%); : NEXT
```

```
SCREEN , , 0, 0    'cambia la pagina attiva nella pagina 0, la pagina
                   'visualizzata
COLOR 14, 1: CLS
LOCATE 3, 28: PRINT " DIMOSTRAZIONE DI PCOPY ";
LOCATE 5, 1: PRINT STRING$(80, "-"); : COLOR 14, 4
LOCATE 24, 1: PRINT SPACE$(80);
PCOPY 0, 2       'copia la pagina visualizzata nella pagina 2
LOCATE 24, 26: PRINT "Premi un tasto qualsiasi per ottenere Aiuto."

r$ = INPUT$(1)
PCOPY 1, 0              'copia lo schermo di Aiuto sulla pagina visualizzata
LOCATE 24, 1: PRINT SPACE$(80):
LOCATE 24, 26: PRINT "Premi un tasto qualsiasi per riprendere."

r$ = INPUT$(1)
PCOPY 2, 0              'ripristina la pagina originale
COLOR 7, 0
END

DATA  69,96,32,84,85,84,84,79,32,78,69
DATA  76,76,65,32,71,85,73,68,65,32
DATA  68,73,32,82,73,70,69,82,73,77
DATA  69,78,84,79,32,65,32,81,66,97,115,105,99
```

PMAP

Si consulti anche

VIEW, WINDOW

Scopo

La funzione PMAP restituisce la coordinata fisica corrispondente alla coordinata logica o viceversa. Se lo schermo contiene un insieme di queste coordinate, è possibile utilizzare questa funzione per convertirle da un sistema a un altro.

Sintassi

```
PMAP(coordinata, n)
```

coordinata è un valore a precisione semplice che rappresenta il valore di una delle coordinate.

n è un valore intero che indica se il parametro coordinata è *x* o *y* e se il relativo valore deve essere convertito dal sistema logico a quello fisico o viceversa; può avere uno dei seguenti valori:

Valore	Descrizione
0	Converte il valore da una coordinata logica a una fisica orizzontale(x)
1	Converte il valore da una coordinata logica a una fisica verticale(y)
2	Converte il valore da una coordinata fisica a una logica orizzontale(x)
3	Converte il valore da una coordinata fisica a una logica verticale(y)

Utilizzo

```
CIRCLE (PMAP(160, 2), PMAP(100, 3)), 50, 3
```

Disegna un cerchio utilizzando le coordinate fisiche per la posizione del centro. Questo garantisce che il cerchio sia sempre centrato all'interno di un viewport di 320 per 200 pixel, indipendentemente dal sistema di coordinate logiche.

Descrizione

PMAP non influenza direttamente i pixel sullo schermo, ma si limita a convertire le coordinate dei pixel da un sistema di coordinate in modo che possano essere utilizzate nelle istruzioni grafiche.

Il sistema di coordinate fisiche viene determinato dal numero di pixel disponibili nella modalità video attiva e dai limiti del viewport grafico definito dall'ultima istruzione VIEW. Se quest'ultima istruzione non è stata eseguita, le dimensioni del viewport sono quelle dello schermo.

L'istruzione WINDOW può essere utilizzata per definire un proprio sistema di coordinate logiche, da assegnare poi al viewport, per poterle poi utilizzare nelle istruzioni grafiche LINE, CIRCLE, PAINT, GET, PUT, PSET e PRESET.

Il valore restituito da PMAP durante la conversione dalle coordinate logiche a quelle fisiche è molto spesso una variabile a precisione semplice, dato che questo tipo di conversione raramente produce un valore di tipo intero.

Il programma di seguito riportato converte un sistema di coordinate logiche, in cui i valori di *x* e *y* variano da -1000 a 1000, in un sistema di coordinate fisiche di 640 per 200 pixel.

```
SCREEN 2
WINDOW (-1000, -1000) - (1000, 1000)
```

Nella Figura 11.8 viene riportata la mappa della conversione dal sistema di coordinate logiche a quello di coordinate fisiche. Si noti che l'origine del primo sistema, ovvero la posizione indicata da (0, 0), è il centro dello schermo e che il valore massimo di *y* si trova nella parte superiore. Nel sistema di coordinate fisiche, l'origine è in corrispondenza dell'angolo superiore sinistro dello schermo, mentre il valore massimo di *y* è nella parte inferiore.

Esempio

L'istruzione WINDOW risulta particolarmente utile perché consente di specificare un sistema di coordinate logiche più piccolo rispetto alle dimensioni fisiche del viewport attivo. Soltanto i punti le cui coordinate si trovano all'interno della struttura logica di una finestra possono essere tracciati, consentendo in questo modo di visualizzare una piccola sezione di quella che potrebbe essere una figura di più ampie dimensioni. Inoltre, modificando l'intervallo delle coordinate logiche è possibile visualizzare altre parti, come se si effettuasse una panoramica con una videocamera.

Generalmente, quando si esegue questa operazione con una videocamera, viene focalizzato un oggetto in movimento su uno sfondo stazionario; lo stesso effetto può essere ottenuto in QBasic, mantenendo fisso l'oggetto e spostando invece lo sfondo. Per realizzare tutto ciò, è sufficiente utilizzare PMAP per tracciare l'immagine di primo

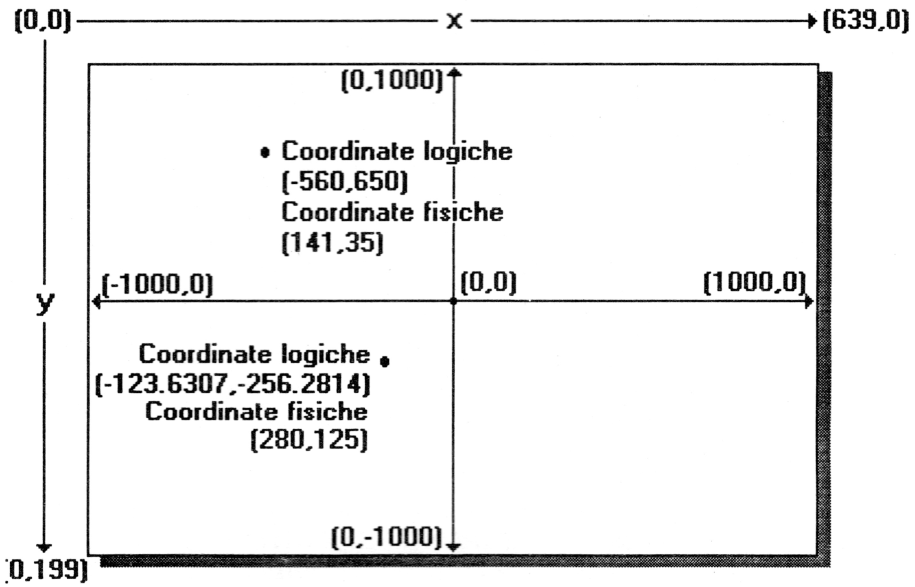

Figura 11.8 PMAP converte le coordinate tra il sistema di coordinate logiche e quello di coordinate fisiche

piano utilizzando le coordinate fisiche, in modo che essa appaia nella stessa posizione nel viewport, indipendentemente dalla parte visibile sullo sfondo. Il programma di seguito riportato utilizza la funzione PMAP per convertire le coordinate fisiche in logiche, in modo che ognuna delle cinque finestre abbia a fuoco l'immagine di primo piano.

```
DECLARE SUB DisegnaMondo ()

SCREEN 9: LINE (0, 0)-(639, 349), 9, BF
x1 = 100: x2 = 256: y1 = 10: y2 = 70 'definisce le coordinate per
                                     'l'istruzione WINDOW
x% = 10: y% = 10                     'definisce le coordinate per il viewport
xCambia% = -100: numIter% = 1
DO
  VIEW (x%, y%)-(x% + 300, y% + 155), 0, 15     'definisce un viewport
  WINDOW SCREEN (x1, y1)-(x2, y2)               'definisce una finestra
  DisegnaMondo                                  'disegna l'immagine di fondo
  numIter% = numIter% + 1
  xFisic = PMAP(150, 2): yFisic = PMAP(75, 3)
  IF y1 > 10 THEN                              'disegna l'immagine di primo piano
     CIRCLE (xFisic, yFisic), 10, 13, -.8, -5.78
  ELSE
     CIRCLE (xFisic, yFisic), 10, 13, -3.9, -2.2
  END IF
  PAINT (PMAP(150, 2), PMAP(68, 3)), 13, 13
  CIRCLE (PMAP(150, 2), PMAP(68, 3)), 1, 1, , , 3
  x% = x% + 320
  IF numIter% = 3 THEN
      x% = 10: y% = 175
```

```
        x1 = x1 - 100: x2 = x2 - 100: xCambia = -xCambia
        y1 = y1 + 22: y2 = y2 + 22
    END IF
    x1 = x1 + xCambia: x2 = x2 + xCambia
    r$ = INPUT$(1)
    LOOP UNTIL numIter% = 5
    VIEW (160, 97)-(460, 252), 0, 15
    WINDOW SCREEN (0, 0)-(256, 120): DisegnaMondo       'disegna l'intero fondo
    CIRCLE (138, 84), 10, 13, -3.9, -2.2     'disegna l'immagine di primo piano
    PAINT (138, 86), 13, 13
    CIRCLE (138, 81), 1, 1, , 3
    END


SUB DisegnaMondo STATIC
    LINE (0, 0)-(256, 27), 14, BF: LINE (0, 28)-(256, 118), 4, B
    LINE (4, 30)-(252, 116), 4, B: LINE (30, 50)-(251, 52), 4, B
    PAINT (128, 51), 4, 4: LINE (3, 72)-(226, 74), 4, B
    PAINT (128, 73), 4, 4: LINE (30, 94)-(251, 96), 4, B
    PAINT (128, 95), 4, 4: PAINT (128, 29), 4, 4
    PAINT (2, 80), 4, 4: PAINT (254, 70), 4, 4
    LINE (208, 27)-(251, 30), 14, BF
    CIRCLE (118, 84), 10, 6: PAINT (118, 84), 6, 6
    CIRCLE (114, 82), 4, 0: CIRCLE (122, 82), 4, 0
    PAINT (114, 82), 15, 0: PAINT (122, 82), 15, 0
    CIRCLE (117, 82), 2, 1: CIRCLE (119, 82), 2, 1
    CIRCLE (118, 86), 2, 0: PAINT (118, 86), 0, 0
END SUB
```

POINT

Si consulti anche

PALETTE, PMAP, PSET, WINDOW

Scopo

La funzione POINT restituisce il colore o il valore di una delle coordinate di un pixel. Se viene richiamata con una coppia di coordinate, restituisce il numero dell'attributo di colore del pixel nella posizione specificata; se viene invece richiamata con un solo argomento, restituisce la coordinata *x* o *y* del cursore grafico sotto forma di coordinata fisica o logica.

Sintassi

```
POINT(x,y)
```

oppure

```
POINT(n)
```

x e y sono le coordinate orizzontali o verticali del pixel da esaminare.

n è un valore intero nell'intervallo da 0 a 3, che indica la coordinata del cursore da restituire; può avere uno dei valori di seguito riportati:

Valore	Descrizione
0	Restituisce la coordinata fisica orizzontale (x)
1	Restituisce la coordinata fisica verticale (y)
2	Restituisce la coordinata logica orizzontale (x)
3	Restituisce la coordinata logica verticale (x)

Utilizzo

```
PRINT POINT(319, 174)
```

Stampa il numero di attributo del pixel posto al centro di uno schermo di 640 per 350 pixel.

```
ultimoY! = POINT(3)
```

Assegna a ultimoY! la coordinata *y* del cursore nel sistema di coordinate logiche definito dall'ultima istruzione WINDOW.

Descrizione

La funzione POINT può essere utilizzata per restituire il numero di attributo di un determinato pixel. Si noti che il numero del colore visualizzato potrebbe essere diverso, se è stata utilizzata l'istruzione PALETTE per riassegnare i numeri di attributo ad altri colori. Se le coordinate fornite indicano un punto all'esterno del viewport attivo, POINT restituisce il valore -1.

Se questa funzione viene richiamata con un solo argomento, restituisce una delle coordinate del cursore grafico, che può essere la coordinata *x* o quella *y* in un sistema di coordinate logiche o fisiche.

Se nessuna istruzione WINDOW è attiva, la funzione POINT restituisce le coordinate fisiche *x* e *y* del cursore, anche se l'argomento indica le coordinate logiche.

Il cursore grafico non è un oggetto fisico, come il cursore testo, ma è semplicemente un termine che indica l'ultimo punto a cui si è fatto riferimento sullo schermo grafico. Quindi, se viene tracciata una linea tra due punti mediante l'istruzione:

```
LINE (10, 10) - (110, 60), 3
```

il cursore grafico si trova in corrispondenza del punto finale della linea, ovvero la posizione (110, 60). Tuttavia, la successiva istruzione LINE ne modifica nuovamente la posizione. Se viene selezionata una nuova modalità video, oppure viene utilizzato CLS per cancellare lo schermo, il cursore si sposta nella propria posizione di default al centro del viewport attivo.

Il cursore grafico può trovarsi all'esterno dei confini del sistema di coordinate in uso. Sebbene QBasic non tracci le parti di una linea o di un cerchio che si trovano all'esterno del viewport, ne disegna tuttavia i punti finali, in modo che l'oggetto possa essere ritagliato correttamente. In questi casi, il valore restituito da POINT potrebbe essere maggiore rispetto al valore massimo o minimo definiti dal sistema di coordinate in uso.

Esempio

I programmi che disegnano i motivi sfruttano spesso la funzione POINT per verificare il colore di un pixel, prima di modificarlo. L'esempio di seguito riportato utilizza questa procedura quando traccia le linee diagonali per impostare il colore soltanto di quei pixel che sono attualmente spenti, mentre a quelli che sono accessi viene assegnato il colore di sfondo. Il risultato ottenuto è un motivo complicato e mutevole.

```
SCREEN 1: COLOR 0, 1: DEFINT A-Z: RANDOMIZE TIMER
DO
  CLS
  xMin = 0: xMax = 319: yMin = 0: yMax = 119     'definisce i confini
  cl = INT(RND * 3) + 1
  xCambia = SGN((RND * 2) - 1.5): yCambia = SGN((RND * 2) - 1.5)
  xInizio = INT(RND * (xMax - xMin)) + xMin
  yInizio = INT(RND * (yMax - yMin)) + yMin
  DO
     IF xInizio + xCambia > xMax OR xInizio + xCambia < xMin THEN
        xCambia = -xCambia
     END IF
     IF yInizio + yCambia > yMax OR yInizio + yCambia < yMin THEN
        yCambia = -yCambia
     END IF
     xInizio = xInizio + xCambia: yInizio = yInizio + yCambia
     x = xInizio: y = yInizio

     IF POINT(x, y) = 0 THEN                   'controlla il colore del pixel
        LINE (x, y)-(x + 2, y + 3), cl, BF
        x = xMax - xInizio: LINE (x, y)-(x + 2, y + 3), cl, BF
        y = yMax - yInizio: LINE (x, y)-(x + 2, y + 3), cl, BF
        x = xInizio: LINE (x, y)-(x + 2, y + 3), cl, BF
     ELSE
        LINE (x, y)-(x + 2, y + 3), 0, BF
        x = xMax - xInizio: LINE (x, y)-(x + 2, y + 3), 0, BF
        y = yMax - yInizio: LINE (x, y)-(x + 2, y + 3), 0, BF
        x = xInizio: LINE (x, y)-(x + 2, y + 3), 0, BF
     END IF
     r$ = INKEY$
     cl = cl + 1: IF cl > 3 THEN cl = 1
     LOOP WHILE r$ = ""
LOOP UNTIL r$ = CHR$(27)
SCREEN 0: WIDTH 80: COLOR 15, 0: CLS
END
```

PRESET, PSET

Si consulti anche

COLOR, SCREEN (Istruzione)

Scopo

Le istruzioni PRESET e PSET impostano o ripristinano un singolo punto o un pixel sullo schermo. Generalmente vengono utilizzate all'interno di una struttura di ciclo di un programma per tracciare grafici e figure geometriche.

Sintassi

```
PRESET [STEP] (x,y) [,colore]
PSET [STEP] (x,y) [,colore]
```

x e **y** sono le coordinate del pixel che si vuole illuminare.

colore è un attributo di colore che definisce il colore che viene assegnato al pixel.

La parola chiave **STEP** indica che le coordinate specificate sono relative alla posizione attuale del cursore grafico.

Utilizzo

```
PRESET (xPos%, yPos%)
```

Sposta il cursore in corrispondenza della coordinata indicata dalle variabili **xPos%** e **yPos%** e assegna al pixel il colore di sfondo in uso.

```
PSET (160, 100) 3
```

Imposta il pixel nella posizione indicata dalle coordinate 160 e 100 al colore assegnato all'attributo 3 nella palette in uso.

Descrizione

L'unica differenza tra **PRESET** e **PSET** consiste nel fatto che, se il parametro dell'attributo viene omesso, **PRESET** imposta un pixel con il colore di sfondo, mentre **PSET** utilizza il colore di primo piano. Gli attributi di colore disponibili dipendono dalla modalità video attiva.

QBasic non visualizza un messaggio di errore se le coordinate specificate si trovano all'esterno del viewport, ma si limita a non eseguire alcuna operazione.

Esempio

Raramente è necessario impostare soltanto un pixel, ma talvolta la forma della figura da tracciare viene determinata da calcoli effettuati durante l'esecuzione del programma ed è troppo complessa per le istruzioni **LINE** e **CIRCLE**. Il programma di seguito riportato accetta i dati immessi e li utilizza per tracciare una figura geometrica.

Il programma sfrutta il numero di gradi specificato dall'utente per tracciare una serie di punti che formano una spirale. L'utente deve anche scegliere tra punti e linee; se vengono indicate queste ultime, il programma utilizza **PSET** per collegare i punti della spirale. Dopo averla tracciata, il programma utilizza **PRESET** per cancellarla.

```
CLS
INPUT "Immettere l'angolo tra i punti in gradi: ", gradi
PRINT : INPUT "<P>unti o <L>inee"; s$
s$ = UCASE$(LEFT$(s$, 1))

xMax = 319: yMax = 119: SCREEN 1      'limiti dello schermo
CLS
x1 = xMax / 2: y1 = yMax / 2          'imposta x1 e y1 (limiti) a metà schermo
PRINT gradi; "GRADI"
radianti = gradi / 57.29578            'converte i gradi in radianti
IF s$ = "P" THEN PRINT "Punti" ELSE PRINT "Linee"

FOR nero% = 0 TO 1                     'prima disegna i punti poi li cancella
```

```
    FOR angolo = 0 TO yMax / 2 STEP radianti
        x = angolo * COS(angolo) + (xMax / 2)
        y = angolo * SIN(angolo) + (yMax / 2)
        IF x < 0 OR x > xMax OR y < 0 OR y > yMax THEN
           EXIT FOR
        END IF
        r$ = INKEY$: IF r$ = CHR$(27) THEN EXIT FOR
        IF s$ = "P" THEN
           IF nero% THEN
              PRESET (x, y)              'cancella i punti
           ELSE
              PSET (x, y)                'disegna i punti
           END IF
        ELSE                             'disegna (o cancella) le linee
           IF x = x1 THEN
                IF y1 < y THEN cambia = 1 ELSE cambia = -1
                FOR z = y1 TO y STEP cambia
                    IF nero% THEN
                       PRESET (x1, z)        'cancella le linee
                    ELSE
                       PSET (x1, z)          'disegna le linee
                    END IF
                NEXT z
           ELSE
                inclinazione = (y - y1) / (x - x1)
                IF ABS(inclinazione) > 1 THEN cambia = ABS(1 / inclinazione)
                ELSE cambia = 1
                IF x1 > x THEN cambia = -cambia
                FOR z = x1 TO x STEP cambia
                    IF nero% THEN
                       PRESET (z, inclinazione * (z - x1) + y1)     'cancella
                                                                    'le linee
                    ELSE
                       PSET (z, inclinazione * (z - x1) + y1)       'disegna
                                                                    'le linee
                    END IF
                NEXT z
                x1 = x: y1 = y
           END IF
        END IF
    NEXT angolo
NEXT nero%
END
```

PUT (Grafica)

Si consulti anche

GET (Grafica), PSET, WINDOW

Scopo

L'istruzione PUT copia nella posizione dello schermo indicata un'immagine grafica da un array in cui è stata memorizzata. La corrispondente istruzione GET copia un'area dello schermo in un array. Queste istruzioni consentono di spostare un'immagine da una parte dello schermo in un'altra.

Sintassi

```
PUT [STEP](x,y), array[(indice)] [,azione]
```

x e y sono le coordinate del vertice superiore sinistro dell'area rettangolare in cui viene copiata l'immagine. La parola chiave STEP specifica che la posizione a cui le coordinate fanno riferimento è relativa alla posizione del cursore grafico.

array è il nome dell'array numerico che contiene i dati dell'immagine. È possibile specificare l'elemento di array in cui inizia la copia, inserendo il valore di indice tra parentesi; se questo valore viene omesso, PUT inizia a copiare l'immagine partendo dal primo elemento dell'array.

azione è una costante che indica in che modo l'immagine deve essere visualizzata sullo schermo, nel caso sia presente un'altra immagine; se questo parametro non viene fornito, PUT utilizza per default XOR. L'elenco di seguito riportato illustra queste costanti.

- AND esegue un'operazione logica AND con ogni pixel nel rettangolo di destinazione e il corrispondente pixel nell'array sorgente. PUT scrive il valore risultante sullo schermo. Se i bit corrispondenti dei pixel sorgente e di quelli di destinazione sono identici, il bit risultante rimane invariato; se entrambi i bit sono diversi da 1, il bit risultante è 0, come illustrato nel seguente esempio:

	Sorgente				
	0	1	1	0	= 6 (Marrone)
	Destinazione				
AND	0	0	1	1	= 3 (Ciano)
	Risultato				
	0	0	1	0	= 2 (Verde)

- OR esegue un'operazione logica OR con ogni pixel nel rettangolo di destinazione e il pixel equivalente nell'array sorgente. PUT scrive il valore risultante sullo schermo. Se il valore di uno dei bit corrispondenti nel pixel è 1, il bit risultante è 1, come illustrato nel seguente esempio:

	Sorgente				
	0	1	1	0	= 6 (Marrone)
	Destinazione				
OR	0	0	1	1	= 3 (Ciano)
	Risultato				
	0	1	1	1	= 7 (Grigio)

- PRESET sovrappone al rettangolo di destinazione un'immagine memorizzata con inversione dei colori cancellando quella esistente, come illustrato nel seguente esempio:

Sorgente

0	1	1	0	= 6 (Marrone)

Risultato

1	0	0	1	= 9 (Blu chiaro)

- PSET sovrappone al rettangolo destinazione l'immagine memorizzata, cancellando quella precedente.

- XOR esegue un'operazione logica XOR (or esclusivo) con ogni pixel nell'array sorgente e il corrispondente pixel di destinazione. PUT scrive il valore risultante sullo schermo. Se il valore di uno, ma non di entrambi i bit corrispondenti nel pixel è 1, il bit risultante è 1, come illustrato nel seguente esempio:

Sorgente

	0	1	1	0	= 6 (Marrone)

Destinazione

XOR	0	0	1	1	= 3 (Ciano)

Risultato

	0	1	0	1	= 5 (Magenta)

Si noti che utilizzando due volte PUT in successione con XOR, il contenuto originale dell'area rettangolare di destinazione viene ripristinato. Questa tecnica risulta utile per la creazione di effetti di animazione in cui il movimento di un'immagine non deve cancellare lo sfondo.

Utilizzo

```
PUT (160, 100), immagine%
```

Copia l'immagine memorizzata nell'array di tipo intero immagine% nell'area rettangolare il cui vertice superiore sinistro si trova nella posizione (160, 100). PUT esegue operazioni logiche XOR con l'immagine e la visualizzazione esistente.

```
PUT STEP(xInc, yInc), immagine(0, 5), PSET
```

Definisce un rettangolo il cui vertice superiore sinistro viene posizionato in base al valore di pixel di xInc sull'asse orizzontale e a quello di yInc sull'asse verticale, partendo dalla posizione del cursore grafico. L'immagine memorizzata nella quinta colonna dell'array a due dimensioni immagine viene copiata in quest'area, cancellando il contenuto precedente.

Descrizione

L'istruzione GET consente di copiare i pixel da un'area rettangolare dello schermo in un array numerico. PUT consente di copiare nuovamente i pixel sullo schermo in qualsiasi punto.

QBasic consente di indicare un operatore logico con l'istruzione PUT, in modo che sia possibile associare l'immagine memorizzata con i grafici già presenti oppure cancellare con quella nuova il contenuto precedente. Ad esempio, è possibile invertire i colori dell'immagine originale (PRESET), sovrapporla ai pixel esistenti (OR) oppure invertire ogni pixel dello schermo che corrisponde a un pixel nell'array sorgente (XOR).

XOR può essere utilizzato anche quando si desidera creare un effetto di animazione, infatti utilizzando PUT due volte con la stessa immagine nello stesso punto, l'immagine viene tracciata e quindi cancellata, ripristinando in questo modo l'area dello schermo al proprio stato originale. È possibile fare in modo che un'immagine sembri essere in movimento su uno sfondo costante, utilizzando PUT due volte con XOR e quindi ripetendo il processo alla posizione successiva nella sequenza.

Avvertenze

Generalmente le coordinate *x* e *y* specificate nell'istruzione PUT definiscono il vertice superiore sinistro dell'area rettangolare in cui viene copiato il contenuto dell'array. Tuttavia, se l'istruzione WINDOW viene utilizzata senza la parola chiave SCREEN per definire un proprio sistema di coordinate logiche, l'asse verticale della visualizzazione viene invertito e le coordinate x e y indicano il vertice inferiore sinistro del rettangolo.

Esempio

PUT può essere utilizzata per copiare le immagini memorizzate dall'istruzione GET in aree successive dello schermo, creando in questo modo un effetto di animazione. Il programma di seguito riportato realizza questo effetto e consente di utilizzare i tasti di direzione per "far camminare" un paio di orme sullo schermo. Inoltre, il programma illustra in che modo le immagini possono essere memorizzate in array multidimensionali oppure copiate da essi.

La prima parte del programma crea un insieme di immagini partendo da una fondamentale. Si noti che l'istruzione DRAW ruota la figura originale per produrre le immagini che puntano in quattro direzioni (corrispondenti ai quattro tasti).

Il programma utilizza il ciclo della seconda parte per controllare la sequenza di animazione. Due istruzioni PUT che utilizzano XOR vengono eseguite in ogni posizione, una per tracciare l'immagine e una per cancellarla; questo garantisce che lo sfondo venga ripristinato, prima che il programma sposti l'immagine nella nuova posizione.

```
DIM pSu%(25), pGiu%(25), pSin%(25), pDes%(25)
DIM piedi%(50, 3, 1)
DIM xInc%(3), yInc%(3)          'array per incrementare la posizione
DIM avanti%(8)                  'array per la figura con il piede in avanti
FOR i% = 0 TO 3: READ xInc%(i%), yInc%(i%): NEXT
frecciaSu$ = CHR$(0) + CHR$(72): frecciaGiu$ = CHR$(0) + CHR$(80)
frecciaSin$ = CHR$(0) + CHR$(75): frecciaDes$ = CHR$(0) + CHR$(77)
piede$ = "C2 S4 R2 D2 L2 U2 BM+1,+1P2,2BM-1,+3R2 D1 L2 U1 BM+1,+1P2,2"
```

```
SCREEN 1: x% = 50: y% = 50
'disegna un piede per ogni direzione e realizza l'immagine
DRAW "BM52,51" + piede$
GET (x% + 2, y% + 1)-(x% + 4, y% + 6), pSu%
DRAW "BM104,56 TA180" + piede$
GET (x% + 52, y% + 1)-(x% + 54, y% + 6), pGiu%
DRAW "BM152,54 TA90" + piede$
GET (x% + 102, y% + 2)-(x% + 109, y% + 4), pSin%
DRAW "BM208,52 TA270" + piede$
GET (x% + 152, y% + 2)-(x% + 159, y% + 4), pDes%

'visualizza il secondo piede per ogni direzione
'e mostra la coppia con il piede opposto in avanti
FOR i% = 1 TO 3
    READ xCambia%, yCambia%
    PUT (x% + xCambia%, y% + yCambia%), pSu%, PSET
    READ xCambia%, yCambia%
    PUT (x% + xCambia%, y% + yCambia%), pGiu%, PSET
    READ xCambia%, yCambia%
    PUT (x% + xCambia%, y% + yCambia%), pSin%, PSET
    READ xCambia%, yCambia%
    PUT (x% + xCambia%, y% + yCambia%), pDes%, PSET
NEXT i%

'ottiene l'immagine per ognuna delle otto paia di piedi
GET (52, 50)-(59, 59), piedi%(0, 0, 0)    'fronte superiore del piede sinistro
GET (102, 50)-(109, 59), piedi%(0, 1, 0) 'fronte inferiore del piede sinistro
GET (151, 51)-(162, 57), piedi%(0, 2, 0) 'fronte sinistro del piede destro
GET (201, 51)-(212, 57), piedi%(0, 3, 0) 'fronte destroe del piede destro
GET (52, 100)-(59, 109), piedi%(0, 0, 1) 'fronte superiore del piede destro
GET (102, 100)-(109, 109), piedi%(0, 1, 1) 'fronte inferiore del piede destro
GET (151, 101)-(162, 107), piedi%(0, 2, 1)'fronte sinistro del piede sinistro
GET (201, 101)-(212, 107), piedi%(0, 3, 1)  'fronte destro del piede sinistro

'realizza l'animazione
CLS : x% = 160: y% = 100
freccia$ = frecciaSu$ + frecciaGiu$ + frecciaSin$ + frecciaDes$
PUT (x%, y%), piedi%(0, 0, 0), XOR
DO
    DO
    tastoPremuto$ = INKEY$
    LOOP WHILE tastoPremuto$ = ""
    muovi% = INSTR(freccia$, tastoPremuto$)
    IF muovi% THEN
         muovi% = ((muovi% + 1) \ 2) - 1   'direzione della figura da muovere
         nuovoX% = x% + xInc%(muovi%): nuovoY% = y% + yInc%(muovi%)
         IF nuovoX% < 10 OR nuovoX% > 300 OR nuovoY% < 10 OR nuovoY% > 180 THEN
             PLAY "T255 L16 O1 G#"                'un piede colpisce il bordo
         ELSE
             'cancella l'immagine precedente
             PUT (x%, y%), piedi%(0, ultimoMov%, avanti%(ultimoMov%)), XOR
             questoMov% = muovi%
             IF questoMov% = ultimoMov% THEN  'cambia il piede che sarà mosso
                IF avanti%(questoMov%) = 0 THEN
                    avanti%(questoMov%) = 1
                ELSE
                    avanti%(questoMov%) = 0
                END IF
             END IF

             'disegna la nuova immagine
             PUT (nuovoX%, nuovoY%), piedi%(0, questoMov%,
             avanti%(questoMov%)), XOR
             ultimoMov% = questoMov%
             x% = nuovoX%: y% = nuovoY%
         END IF
    END IF
```

```
LOOP UNTIL tastoPremuto$ = CHR$(27)
END

DATA 0,-5,0,5,-5,0,5,0
DATA 6,3,56,3,105,5,155,5,2,53,52,53,105,52,155
DATA 52,6,51,56,51,102,55,152,55
```

SCREEN (Funzione)

Si consulti anche

COLOR, LOCATE, SCREEN (Istruzione)

Scopo

La funzione SCREEN restituisce informazioni riguardanti una posizione specificata. Per default, la funzione restituisce il codice ASCII del carattere nella posizione (riga, colonna) indicata; in modalità testo, restituisce anche il numero di attributo del colore di quel carattere.

Sintassi

```
SCREEN (riga,colonna [,flagcolore])
```

riga e colonna sono numeri interi che specificano la posizione del carattere.

flagcolore è un intero facoltativo. Se questo parametro viene tralasciato, la funzione SCREEN restituisce il codice ASCII del carattere nella posizione indicata; se viene inserito un valore diverso da 0, SCREEN restituisce il numero di attributo del colore del carattere.

Utilizzo

```
PRINT SCREEN(5, 10)
```

Stampa il codice ASCII del carattere visualizzato sullo schermo nella riga 5, colonna 10.

```
clr% = SCREEN(riga%, colonna%, 1)
```

Assegna alla variabile di tipo intero clr% il numero di attributo del colore del carattere alla posizione specificata da riga% e colonna%

Descrizione

La funzione SCREEN è la versione in modalità testo della funzione grafica POINT e restituisce informazioni relative a un carattere presente nella posizione specificata sullo schermo. Questa funzione restituisce il codice ASCII del carattere o il numero dell'attributo di colore del carattere.

Con i sistemi monocromatici, il numero dell'attributo controlla l'intensità del carattere e se quest'ultimo è visualizzato inverso, lampeggiante o sottolineato.

La funzione SCREEN restituisce anche i valori ASCII nelle modalità grafiche; essa restituisce il numero dell'attributo soltanto nella modalità video 0, ovvero quella di testo.

Errori

Se viene utilizzata l'istruzione VIEW PRINT per definire un viewport di testo, la funzione SCREEN opera soltanto all'interno di esso. Se si cerca di leggere un carattere presente su una riga non compresa nel viewport, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita."

Esempio

La funzione SCREEN risulta utile per la lettura dei dati visualizzati sullo schermo dal DOS o da un altro programma. L'esempio di seguito riportato utilizza questa funzione con il comando SHELL per determinare l'etichetta del volume del disco nell'unità specificata.

```
CLS : LOCATE 10, 1
INPUT "Etichetta di volume per quale drive"; drive$
IF drive$ <> "" THEN
    drive$ = " " + LEFT$(drive$, 1) + ":"
END IF
VIEW PRINT 20 TO 21
etichetta$ = ""
SHELL "VOL" + drive$             'chiama la funzione VOL del DOS
FOR i% = 23 TO 33
    etichetta$ = etichetta$ + CHR$(SCREEN(21, i%))      'legge l'etichetta
NEXT i%
VIEW PRINT: LOCATE 22, 1
PRINT "L'etichetta che la funzione VOL ha restituto per "; etichetta$
'visualizza l'etichetta
END
```

SCREEN (Istruzione)

Si consulti anche

COLOR, PALETTE, PCOPY, VIEW, WIDTH,(Schermo). WINDOW

Scopo

L'istruzione SCREEN definisce diverse caratteristiche dello schermo, consentendo di selezionare diverse modalità video, ognuna delle quali controlla il numero di pixel, i colori e le pagine video disponibili.

Sintassi

```
SCREEN [modalità] [,[colore] [,[paginaattiva] [,paginavisualizzata]]]
```

modalità è un numero nell'intervallo da 0 a 13 che indica la modalità di visualizzazione. QBasic non supporta le modalità 5 e 6.

colore è un'espressione numerica che quando è vera (diversa da zero) disattiva la visualizzazione dei colori sui computer collegati a monitor composti o alla televisione a colori. Questo parametro viene ignorato in tutte le modalità, tranne che nella 0 e nella 1.

paginaattiva è un intero che indica la pagina video in cui viene stampato l'output, mentre paginavisualizzata è un intero che specifica la pagina video attualmente visualizzata. I due interi devono essere valori diversi.

Utilizzo

```
SCREEN 9, , 1, 0
```

Seleziona la modalità a 16 colori ad alta risoluzione (640 per 350 pixel) disponibile con gli adattatori EGA e VGA. Tutte le istruzioni grafiche successive vengono tracciate sulla pagina video 0; la pagina 1 diventa quella visualizzata. Si noti che, sebbene l'argomento colore non sia valido in questa modalità, è necessario utilizzare una virgola per mantenerne la posizione nell'elenco dei parametri.

```
SCREEN 0
```

Disattiva l'output grafico e riporta alla modalità video 0, ovvero quella di testo.

Descrizione

Le modalità video disponibili vengono determinate dal tipo di adattatore installato sul computer. QBasic supporta, oltre a tutti quelli generalmente utilizzati con i PC IBM (XT, AT, PS/2) e compatibili, anche gli adattatori dei sistemi Olivetti e AT&T. Nella Tabella 11.13 sono elencati questi adattatori e le modalità video che possono utilizzare; si noti che molti di questi adattatori sono compatibili con i propri predecessori e supportano tutte le modalità video precedenti.

Alcuni adattatori possono memorizzare contemporaneamente più di una pagina di informazioni dello schermo; queste, definite pagine video, vengono numerate da 0 fino al numero massimo disponibile. Non soltanto è possibile utilizzare SCREEN per selezionare le pagine attive e visibili, ma è anche possibile copiare il contenuto dello schermo da una pagina a un'altra con l'istruzione PCOPY. Nella Tabella 11.14 è elencato il numero massimo di pagine video disponibili in ogni modalità per i tipi diversi di adattatori.

Tabella 11.13 Gli adattatori video e le modalità video disponibili

Adattatore	Modalità video disponibili
MDPA (Monochrome Display and Printer Adapter)	0
Olivetti	0, 4
AT&T	0, 4
CGA (Color Graphics Adapter)	0, 1, 2
MCGA (Multi-Color Graphics Array)	0, 1, 2, 11, 13
HGC (Hercules Graphics Card)	0, 3
EGA (Enhanced Graphcis Adapter)	0, 1, 2, 7, 8, 9, 10*
VGA (Video Graphics Array)	0, 1, 2, 7, 8, 9, 10*, 11, 12, 13

* La modalità video 10 è disponibile soltanto se gli adattatori EGA e VGA sono collegati a monitor monocromatici ad alta risoluzione

Tabella 11.14 Il numero di pagine video disponibili in base a ogni modalità video

Schermo	MDPA	CGA	MCGA	HGC	Olivetti	AT&T	EGA	VGA	Osservazioni
0	1	4	8	2	1	1	8	8	80 per 25 testo
	-	8	8	-	-	-	8	8	40 per 25 testo
	-	-	-	-	-	-	8	8	40 per 43 testo
	-	-	-	-	-	-	-	4	40 per 50 testo
	-	-	-	-	-	-	4	4	80 per 43 testo
	-	-	-	-	-	-	-	4	80 per 50 testo
1	-	1	1	-	-	-	1	1	
2	-	1	1	-	-	-	1	1	
3	-	-	-	2	-	-	-	-	Soltanto Hercules
4	-	-	-	-	1	1	-	-	
7*	-	-	-	-	-	-	8	8	
8*	-	-	-	-	-	-	4	4	
9*	-	-	-	-	-	-	2	2	
10*	-	-	-	-	-	-	4	4	
11	-	-	1	-	-	-	-	1	
12	-	-	-	-	-	-	-	1	
13	-	-	1	-	-	-	-	1	

* I valori relativi alle modalità viedo 7, 8, 9 e 10 prevedono che si disponga di un adattattore EGA o VGA con 256 KB di memoria dedicata

Modalità video 0

La modalità video 0 è modalità di testo ed è quella di default per tutti gli adattatori. Se si dispone di un MDPA, HGC o di un EGA o VGA con un monitor monocromatico, l'unica dimensione disponibile è quella di 25 righe per 80 colonne. Gli altri formati di testo sono disponibili soltanto con adattatori diversi e con i monitor a colori, come riportato nella Tabella 11.14.

L'istruzione WIDTH (Schermo) può essere utilizzata per modificare il numero di righe e di colonne.

Il testo può essere visualizzato in base a qualsiasi combinazione di 16 attributi di primo piano e 8 di sfondo. Sui monitor monocromatici, i numeri dell'attributo controllano la sottolineatura e due gradi di intensità del carattere; sui monitor a colori, questi numeri determinano il colore del testo e dello sfondo; entrambi supportano i caratteri lampeggianti. I computer che dispongono di un adattatore EGA, VGA o MCGA e di un monitor a colori possono assegnare nuovamente gli attributi di colore, utilizzando l'istruzione PALETTE.

Modalità video 1

La modalità video 1 è una modalità grafica a media risoluzione (320 per 200 pixel) con un formato di testo di 25 righe per 40 colonne. Dato che la televisione e i monitor compositi possono visualizzare sia i grafici che il testo, quando è attiva questa modalità

il colore è abilitato per default. Per disattivarlo, è sufficiente richiamare l'istruzione SCREEN impostando il terzo argomento a un valore diverso da 0.

Con questa modalità sono disponibili 4 colori. Il colore di sfondo può essere uno qualsiasi dei 16 attributi standard elencati nella Tabella 11.4. Per indicare i tre colori di primo piano disponibili, è possibile scegliere tra le due palette di colori riportate nella Tabella 11.5.

Come nelle altre modalità video, è possibile utilizzare l'istruzione PALETTE per assegnare nuovamente gli attributi di colore sui computer che dispongono di un adattatore EGA, VGA o MCGA.

Modalità video 2

La modalità video 2 è una modalità grafica ad alta risoluzione (640 per 200 pixel) con un formato di testo di 25 righe per 80 colonne. Questa modalità supporta soltanto due colori, generalmente il bianco (quello di primo piano) e il nero (quello di sfondo), ma è possibile utilizzare l'istruzione PALETTE per utilizzare i 16 colori standard con i sistemi EGA, VGA o MCGA.

Modalità video 3

La modalità video 3 è una modalità grafica ad alta risoluzione (720 per 348 pixel) con un formato di testo di 25 righe per 80 colonne ed è disponibile soltanto sui computer che dispongono dell'adattatore HGC (Hercules Graphics Card). Per attivarla è necessario eseguire il programma MS-HERC-COM, prima di utilizzare QBasic. La modalità video 3 supporta soltanto i grafici monocromatici e non è possibile assegnare nuovamente i colori di sfondo e di primo piano con l'istruzione PALETTE.

Modalità video 4

La modalità video 4 è una modalità grafica ad alta risoluzione (640 per 400 pixel) con un formato di testo di 25 righe per 80 colonne ed è disponibile soltanto sui Personal Computer Olivetti modello M24, M240, M28, M280, M380, M380/C e M380/T e sulla serie 6300 dei Personal Computer AT&T. Questa modalità supporta 16 colori di primo piano che possono essere selezionati utilizzando l'istruzione COLOR, mentre il colore di sfondo, che è nero, non può essere modificato.

Modalità video 7

La modalità video 7 è disponibile soltanto sui computer che utilizzano un adattatore EGA o VGA ed è simile alla modalità 1, tranne per il fatto che sono disponibili 16 colori, invece di 4. L'istruzione PALETTE può essere utilizzata per assegnare nuovamente gli attributi di colore; questa modalità ha una risoluzione di 320 per 200 pixel e un formato di testo di 25 righe per 40 colonne. Sono disponibili fino a otto pagine video, in base alla quantità di memoria video di cui dispone l'adattatore utilizzato.

Modalità video 8

La modalità video 8 è una versione avanzata della modalità 2 per i computer che utilizzano un adattatore EGA o VGA; ha una risoluzione di 640 per 200 pixel, un

formato di testo di 25 righe per 80 colonne e supporta 16 colori, a cui è possibile assegnare un nuovo attributo con l'istruzione PALETTE. Sono disponibili quattro pagine video per gli adattatori che hanno 256 KB di memoria video.

Modalità video 9

La modalità video 9 è quella grafica standard per i computer con un adattatore EGA, anche se può essere utilizzata con i sistemi VGA; ha una risoluzione di 640 per 350 pixel e un formato di testo di 80 righe per 25 colonne o di 80 righe per 43 colonne.

Il numero di colori e le pagine video disponibili dipendono dalla quantità di memoria video dell'adattatore. Le versioni precedenti dell'EGA comprendevano soltanto 64 KB di memoria, che supportano soltanto una pagina video e 4 colori su una palette di 16. L'adattatore VGA e l'EGA con 256 KB o più di memoria possono passare tra due pagine e visualizzare 16 colori da una palette di 64.

Modalità video 10

La modalità video 10 è disponibile soltanto con i computer che dispongono di un adattatore EGA o VGA collegati a un monitor monocromatico ad alta risoluzione; ha una risoluzione di 640 per 350 pixel e un formato di testo di 80 righe per 25 colonne o di 80 righe per 43 colonne. Sono disponibili quattro pagine video per gli adattatori con una memoria video di 256 KB.

Questa modalità supporta quattro attributi, ognuno dei quali supporta uno tra 9 pseudocolori per il controllo di intensità e lampeggio.

Modalità video 11

La modalità video 11 è disponibile soltanto con i computer che dispongono di un adattatore VGA o MCGA; ha una risoluzione di 640 per 480 pixel e un formato di testo di 80 righe per 30 colonne o di 80 righe per 60 colonne, mentre non è disponibile il formato di 80 righe per 25 colonne.

Sono disponibili soltanto due attributi di visualizzazione (uno per il colore di primo piano e uno per quello di sfondo), ma a ognuno di questi può essere assegnato uno dei 262144 colori con l'istruzione PALETTE.

Modalità video 12

La modalità video 12 è disponibile soltanto con i computer che utilizzando un adattatore VGA; ha una risoluzione di 640 per 480 pixel e un formato di testo di 80 righe per 30 colonne o di 80 righe per 60 colonne. Al contrario della modalità 11, tuttavia, supporta 16 attributi di colore da una palette composta di 262144 colori.

Modalità video 13

La modalità video 13 è una modalità a media risoluzione (320 per 200 pixel) con un formato testo di 40 righe per 25 colonne ed è disponibile sia con l'adattatore VGA che MCGA. In questa modalità, i grafici possono essere tracciati sfruttando uno dei 256 attributi della palette composta da 262144 colori.

Note

Nelle modalità grafiche, la modifica del numero delle colonne di caratteri sullo schermo con l'istruzione WIDTH (Schermo) ottiene lo stesso effetto della modifica apportata al numero della modalità video. Ad esempio, l'esecuzione dell'istruzione WIDTH 40 in modalità 2 fa passare immediatamente alla modalità video1.

Se viene selezionata una modalità video diversa da quella attiva QBasic cancella lo schermo, proprio come se fosse stata eseguita un'istruzione CLS. Tuttavia, se l'istruzione SCREEN viene utilizzata per selezionare un'altra pagina, senza modificare la modalità, QBasic non elimina i grafici o il testo dalla pagina video, questi però potrebbero non essere più visibili, se la pagina specificata non è la stessa.

Esempio

Per esaminare un programma che illustra l'istruzione SCREEN, si consulti l'esempio riportato alla voce COLOR.

VIEW

Si consulti anche

PMAP, SCREEN (Istruzione), VIEW PRINT, WINDOW

Scopo

L'istruzione VIEW definisce un'area rettangolare dello schermo, delimitando una zona in cui devono essere tracciati tutti i grafici. Questa area viene definita viewport grafico. Se il programma creato utilizza sia testo che grafici, è bene definire un viewport grafico, in modo da mantenere separati i due tipi di output, impedendo alle figure di sovrascrivere il testo visualizzato.

Sintassi

```
VIEW [SCREEN] (x1,y1)-(x2,y2) [,[colore] [,bordo]]]
```

La parola chiave SCREEN controlla le coordinate che possono essere utilizzate nelle istruzioni grafiche successive che, in questo caso, continuano a utilizzare il vertice superiore sinistro dello schermo fisico come origine. Se questa parola chiave viene omessa, tutte le istruzioni grafiche successive assumono come origine il vertice superiore sinistro del viewport, valore che viene specificato da (0, 0).

x1, y1, x2 e y2 sono numeri interi che indicano le coordinate fisiche dello schermo relative ai vertici opposti del viewport in fase di definizione. x1 e x2 possono variare da 0 fino alla massima risoluzione orizzontale, mentre y1 e y2 possono variare da 0 fino alla massima risoluzione verticale della modalità video attiva.

colore è un'espressione di tipo intero che indica il colore di sfondo; utilizzando questo parametro, il viewport viene cancellato e impostato in base al colore specificato, se invece colore non viene inserito, il viewport non viene eliminato e i grafici o il testo presente all'interno continuano a essere visualizzati.

bordo è un'espressione di tipo intero che, quando è vera (un valore diverso da 0), traccia attorno al viewport una linea del colore indicato (se il parametro bordo viene utilizzato, senza però specificare un attributo, è necessario inserire la prima virgola per mantenere la posizione di colore nell'elenco dei parametri).

Se l'istruzione VIEW viene utilizzata senza alcun argomento, qualsiasi istruzione VIEW precedente viene annullata e tutto lo schermo viene definito come il viewport grafico in uso. Tuttavia, VIEW non cancella lo schermo né modifica la visualizzazione.

Utilizzo

```
VIEW SCREEN (50, 20)-(150, 100)
```

Definisce le coordinate fisiche specificate come i nuovi limiti del viewport grafico. Dato che è stata inserita la parola chiave SCREEN, l'output dei grafici successivi utilizza le coordinate assolute la cui origine si trova nel vertice superiore sinistro dello schermo fisico.

Descrizione

L'istruzione VIEW consente di definire l'area dello schermo in cui vengono visualizzati i grafici e può anche essere utilizzata per modificare la visualizzazione del sistema di coordinate. Nelle istruzioni grafiche, le coordinate fisiche si riferiscono per default al vertice superiore sinistro dello schermo come origine, orizzontalmente da sinistra a destra (asse *x*) e verticalmente dall'alto verso il basso (asse *y*). Questo sistema di coordinate fisiche rimane invariato se l'istruzione VIEW viene utilizzata con la parola chiave SCREEN; in caso contrario, VIEW modifica la posizione dell'origine, che prende come riferimento il vertice superiore sinistro del viewport.

Questa istruzione non visualizza i grafici presenti all'esterno del viewport specificato, ma soltanto quelli che si trovano all'interno di quest'area. Una figura che non è completamente racchiusa in un viewport viene definita una figura "ritagliata".

In un programma è possibile utilizzare diverse istruzioni VIEW, anche se i grafici vengono visualizzati soltanto nel viewport attivo al momento, ovvero l'area definita dall'ultima istruzione VIEW. Dato che le coordinate specificate da questa istruzione vengono sempre interpretate come coordinate fisiche dello schermo, non è necessario mantenere le nuova coordinate all'interno del viewport attivo.

Se VIEW viene utilizzata per inizializzare un viewport grafico, l'istruzione CLS cancella soltanto il contenuto di quel viewport. Tuttavia, se il programma esegue successivamente un'istruzione VIEW PRINT, il viewport di testo definito diventa l'area attiva e una chiamata a CLS cancella soltanto quest'ultimo viewport.

Esempio

Il programma di seguito riportato crea un grafico relativo ai guadagni e alle perdite di una società immaginaria. Viene utilizzata VIEW per creare un viewport per le etichette del grafico e un altro per il contenuto del grafico.

```
SCREEN 9: LINE (0, 20)-(639, 349), 1, BF

VIEW (120, 180)-(510, 330), 0, 15     'definisce il viewport per le etichette
                                      'del grafico
WINDOW (0, -20000)-(12, 20000)
```

```
LOCATE 14, 33: PRINT "Guadagni e Perdite"
LOCATE 15, 35: PRINT "(in migliaia)"
LOCATE 14, 18: PRINT "20000"
LOCATE 19, 16: PRINT "0": LOCATE 23, 17: PRINT "-20000"
LINE (1, 0)-(12, 0), 15: LINE (1, -14000)-(1, 16000)

VIEW (152, 180)-(510, 330) 'definisce il viewport per i contenuti del grafico
WINDOW (1, -20000)-(12, 20000)
PSET (0, 0)
RESTORE
FOR i% = 1 TO 12
    READ perdite!: LINE -(i%, perdite!), 15    'ottiene i dati per il grafico
NEXT i%

PAINT (3, 2000), 14, 15: PAINT (10, -3000), 12, 15      'disegna il grafico
PAINT (12, 800), 14, 15: VIEW: WINDOW
LOCATE 23, 30: PRINT "Guadagno"
LINE (300, 310)-(320, 320), 14, BF
LOCATE 23, 46: PRINT "Perdite"
LINE (420, 310)-(440, 320), 12, BF
END


DATA  4013.28,7269.88,11152.75,9327.08,4148.20,828.74
DATA  -159.15,-788.61,-2693.80,-5164.02,-1176.53,1297.63
```

WINDOW

Si consulti anche

PMAP, VIEW

Scopo

L'istruzione WINDOW deve essere utilizzata per definire un sistema di coordinate logiche di propria scelta per il viewport attivo; può anche essere sfruttata per invertire l'orientamento dall'alto verso il basso della visualizzazione dei grafici, utilizzando invece il sistema cartesiano (con origine nel vertice inferiore sinistro del viewport).

Sintassi

```
WINDOW [[SCREEN] (x1,y1)-(x2,y2)]
```

x1, y1, x2 e y2 sono valori a precisione semplice nell'intervallo da -3.37E+38 a 3.37E+38, che definiscono le dimensioni logiche del viewport attivo. Il valore più piccolo delle due coordinate *x* è il limite sinistro, mentre il valore più grande è il limite destro. Per default, la coordinata *y* che ha il valore minore indica il limite inferiore del viewport, mentre quella che ha il valore maggiore specifica il limite superiore. Se viene invece utilizzata la parola chiave SCREEN, il valore minore della coordinata *y* diventa il limite superiore del viewport.

Se WINDOW viene utilizzata senza argomenti, le istruzioni grafiche non utilizzano più le coordinate logiche, ma per default quelle fisiche del viewport.

Utilizzo

```
WINDOW (88.09, 1.5E+6) - (89.09, 2.5E+6)
```

Definisce le coordinate logiche di un viewport che visualizza un grafico creato sfruttando numeri con valori alti. Si noti che è possibile utilizzare sia numeri decimali che in virgola mobile come coordinate logiche.

Descrizione

L'istruzione WINDOW consente di tradurre dati effettivi in coordinate grafiche, senza dover scalare ogni valore per poterlo inserire nei confini dello schermo fisico. Se il programma creato richiama WINDOW, le dimensioni e la forma dei grafici tracciati appaiono più o meno identici nella modalità video 1 come in quella 9. Le uniche diversità tra le modalità, quando si utilizza WINDOW, riguardano il fatto che i colori si differenziano e che le linee e le curve nella modalità video con risoluzione più bassa sono notevolmente più frastagliate.

Nel sistema di coordinate cartesiano, l'altezza di un oggetto viene misurata dal basso verso l'alto, non dall'alto verso il basso come nel sistema di coordinate fisiche. WINDOW utilizza per default il primo sistema, che però può essere ignorato inserendo l'opzione SCREEN.

Dopo aver definito un sistema di coordinate logiche, le successive istruzioni grafiche, quali LINE, CIRCLE, PAINT, PSET, PRESET, GET e PUT utilizzano i valori di questo sistema come coordinate. QBasic traduce queste coordinate logiche in posizioni fisiche dello schermo, quando deve tracciare un figura. L'istruzione DRAW rappresenta un'eccezione, dato che accetta soltanto le coordinate fisiche dello schermo.

Se il programma creato fa riferimento a punti che si trovano all'esterno rispetto alle dimensioni logiche del viewport, QBasic li traccia, senza però visualizzarli sullo schermo. Il fatto che vengano ugualmente tracciati assicura che qualsiasi linea o curva che passa attraverso questi punti sia correttamente tracciata nelle intersezioni con i limiti dello schermo.

Se un'istruzione grafica traccia una figura in modo che l'ultimo punto disegnato si trovi all'esterno rispetto ai confini del viewport, la posizione del cursore grafico rimane in quel punto, anche se è al di fuori dei limiti del viewport. QBasic utilizza quel punto per tracciare le istruzioni grafiche successive che utilizzano la parola chiave STEP per indicare le coordinate relative.

Esempio

WINDOW può essere utilizzata per effettuare degli zoom su di un disegno, tracciandolo all'interno di sistemi di coordinate sempre più grandi. È anche vero il contrario: la definizione di valori più piccoli per le dimensioni logiche dei viewport successivi consente di ingrandire le immagini contenute. Il programma di seguito riportato sfrutta questa tecnica, tracciando una figura sei volte e utilizzando ogni volta dimensioni più piccole per le coordinate logiche, in modo da ingrandire la figura.

```
DECLARE SUB DisegnaFigura ()

SCREEN 9
LINE (0, 0)-(639, 349), 11, BF
x1 = 0: x2 = 1020: y1 = 0: y2 = 764
xPos% = 10: yPos% = 10: i% = 1

DO
  VIEW (xPos%, yPos%)-(xPos% + 200, yPos% + 155), 0, 15'definisce il viewport
```

```
  WINDOW SCREEN (x1, y1)-(x2 / i%, y2 / i%)  'definisce le coordinate logiche
  DisegnaFigura
  i% = i% + 1
  xPos% = xPos% + 210
  IF i% = 4 THEN xPos% = 10: yPos% = 175
LOOP UNTIL i% = 7
END

DATA 68,4,200,76,52,12,112,44,128,52,172,76,128,52
DATA 68,84,112,44,84,60,128,68,100,84,68,36,96,52
DATA 128,68,154,84,128,68,128,116,130,54,130,68,68
DATA 4,52,12,172,76,142,90,142,76,142,108,142,108
DATA 200,76,200,76,200,92,200,92,68,164,128,116,84
DATA 140,52,12,52,154,52,154,68,164,68,164,68,100
DATA 68,36,68,84,84,45,84,76,84,109,84,140,68,100
DATA 96,116,84,124,112,108,68,84,128,116,85,75,113
DATA 91,112,77,112,108,84,119,92,115,142,86,150,82
DATA 180,66,186,62,186,62,236,90,236,90,68,184,68
DATA 184,16,154,16,154,52,133,16,154,16,160,16,160
DATA 68,190,68,190,68,184,68,190,236,96,236,96,236
DATA 90

SUB DisegnaFigura STATIC
    RESTORE
    FOR i% = 1 TO 40                              'legge le coordinate
        READ x1%, y1%, x2%, y2%: LINE (x1%, y1%)-(x2%, y2%), 1   'disegna
                                                                 'il profilo
    NEXT i%
    'colora la figura
    PAINT (56, 20), 1, 1: PAINT (136, 64), 1, 1
    PAINT (120, 80), 1, 1: PAINT (152, 110), 14, 1
    PAINT (76, 48), 14, 1: PAINT (124, 60), 14, 1
    PAINT (68, 12), 2, 1: PAINT (80, 84), 2, 1
    PAINT (92, 128), 2, 1: PAINT (36, 150), 12, 1
    PAINT (150, 125), 12, 1: PAINT (80, 120), 14, 1
    PAINT (36, 168), 1, 1: PAINT (150, 142), 14, 1
    PAINT (88, 118), 12, 1: PAINT (144, 86), 12, 1
    PAINT (100, 120), 2, 1: PAINT (165, 90), 2, 1
END SUB
```

Capitolo 12

Suono

INTRODUZIONE

La musica e gli effetti sonori possono rendere più interessante qualsiasi programma, sia esso un gioco, un programma educativo o un'applicazione. È sufficiente aggiungere una semplice melodia nello schermo di apertura, un beep come segnale di avvertimento o una tonalità piacevole come segno di apprezzamento per riuscire a ravvivare l'attenzione dell'utente.

Nella Tabella 12.1 sono riportate in ordine alfabetico le istruzioni e le funzioni legate al suono.

UTILIZZO DELLA MUSICA

Prima di decidere che tipo di musica suonare nei programmi, è necessario innanzitutto prendere in considerazione lo scopo del programma che si sta creando. In che modo la musica può aiutare a intrattenere l'utente o a comunicare con lui? Fatta eccezione per quei programmi il cui scopo esplicito è quello di insegnare o illustrare la musica, l'inserimento di una parte musicale ha generalmente due scopi.

Il primo, spesso sfruttato nei giochi, è quello di creare un'atmosfera suonando una musica in sottofondo, mentre il programma esegue altre istruzioni (una melodia potrebbe anche intrattenere un utente, mentre il programma non genera alcun output visivo).

Il secondo scopo della musica è quello di richiamare l'attenzione dell'utente, per dare un suggerimento, un avvertimento o una ricompensa. In un programma educativo, ad esempio, un trionfante suono di tromba potrebbe significare che lo studente ha risposto esattamente a una domanda difficile, oppure una melodia potrebbe corrispondere a un errore. Il programma potrebbe anche suonare un breve motivo che indichi allo studente il momento di rispondere a una domanda o di

Tabella 12.1 Le istruzioni e le funzioni di QBasic relative al suono

Istruzione o funzione	Descrizione
BEEP	Fa emettere un suono all'autoparlante del computer.
ON PLAY GOSUB	Indica la subroutine che aggiunge le note al buffer musicale, quando è attivata la gestione degli eventi di suoni. Le note vengono suonate in sottofondo.
PLAY (Funzione)	Restituisce il numero di note restanti nel buffer musicale.
PLAY (Istruzione)	Suona la melodia specificata, eseguendo i comandi relativi all'ottava, al tono, alla durata, al tempo e alla musica in primo piano e in sottofondo.
PLAY ON/OFF/STOP	Attiva, disattiva o sospende la gestione degli eventi di suono che si verifica quando il buffer musicale contiene un numero inferiore di note rispetto a quello specificato.
SOUND	Genera un tono della frequenza specificata per una durata determinata.

immettere alcune informazioni. Tutto ciò potrebbe risultare utile nei programmi rivolti ai bambini o alle persone con problemi visivi.

Nella maggior parte dei casi la musica deve essere utilizzata con parsimonia, anche perché molti PC dispongono di piccoli autoparlanti che non sono in grado di generare frequenze molto alte o molto basse.

TRADUZIONE DI UNO SPARTITO IN ISTRUZIONI PLAY

Probabilmente non si desidera utilizzare QBasic per comporre musiche proprie, infatti, anche se le diverse istruzioni per suonare non sono difficili da utilizzare, ovviamente non sono così semplici da leggere come uno spartito musicale. É più probabile che si scelga di trascrivere musica scritta nella notazione standard. Nella Figura 12.1 è riportata la melodia di "Red River Valley" scritta su un pentagramma, mentre di seguito sono riportati i comandi utilizzati con l'istruzione PLAY per generare le note corrispondenti.

L'istruzione PLAY esegue i compiti specificati in una stringa di comandi che indicano il tono (ad esempio A, C# o E), la durata, il tempo e altre caratteristiche della musica. Si osservi che è necessario utilizzare per le note la notazione inglese, per cui A=La, B=Si, C=Do, D=Re, E=Mi, F=Fa, G=Sol. Anche se non è obbligatorio utilizzare gli spazi per separare i comandi nella stringa, è meglio farlo per facilitare la lettura. Il programma di seguito riportato utilizza un'istruzione PLAY per ogni battuta della musica di "Red River Valley", in modo da migliorare la leggibilità. Quando si traduce una nota nei comandi PLAY, è necessario indicare sia il tono che la durata.

Red River Valley

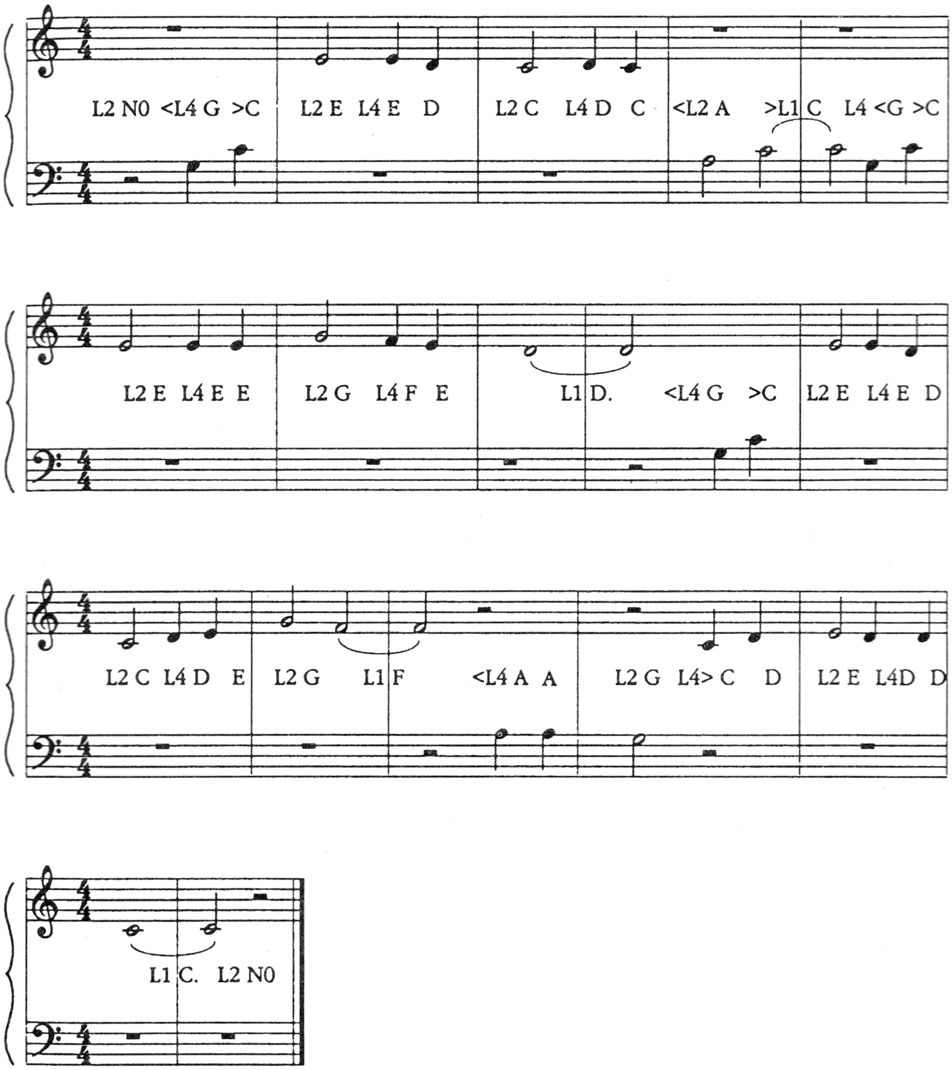

Figura 12.1 La notazione in QBasic e quella musicale standard della canzone "Red River Valley"

```
'l'aria di Red River Valley
PLAY "O3"               'imposta 3 ottave, tonalità C media
PLAY "L2 NO < L4 G > C"
PLAY "L2 E L4 E D"
PLAY "L2 C L4 D C"
PLAY "< L2 A > L1 C L4 < G > C"
PLAY "L2 E L4 E E"
PLAY "L2 G L4 F E"
PLAY "L1 D. < L4 G > C"
PLAY "L2 E L4 E D"
PLAY "L2 C L4 D E"
PLAY "L2 G L1 F < L4 A A"
PLAY "L2 G L4 > C D"
PLAY "L2 E L4 D D"
PLAY "L1 C. L2 NO"
```

Indicazione del tono e della durata

Si esamini il listato del programma che suona la musica di "Red River Valley". La prima istruzione (PLAY "O3") imposta l'ottava (il comando inizia con la lettera O, non con il numero 0). La musica può essere suonata in una qualsiasi delle sette ottave numerate da 0 a 6, dove 0 corrisponde all'ottava più bassa, 3 all'ottava contenente il C centrale e 6 all'ottava più alta.

I comandi contenuti nella stringa della seconda istruzione PLAY indicano le seguenti azioni:

L2 NO	Suona una pausa (nota 0) per 2 battiti (la lunghezza di una nota di 1 mezzo).
<	Abbassa di un'ottava la nota successiva
L4 G	Suona le note G per 1 battito (la lunghezza di un nota di un quarto).
>	Alza di un'ottava la nota seguente.
C	Suona la nota C per 1 battito (la lunghezza di una nota di 1 quarto).

Generalmente, il metodo più conveniente è quello di specificare il tono delle note utilizzando le lettere standard, da A a G. Tuttavia, è anche possibile sfruttare il comando N per specificare il numero assoluto di nota: da N0 a N84. Ad esempio N30 suona la trentesima nota nell'intervallo di sette ottave. Nella Tabella 12.2 sono elencate le note e i numeri corrispondenti. N0 indica una pausa, che può anche essere specificata mediante il comando P, seguito da un numero che ne indica la durata. Ad esempio, P4 indica una pausa di 1 quarto.

Sebbene l'esempio di "Red River Valley" non utilizzi i diesis e i bemolle per alzare o abbassare i toni, QBasic consente di farlo inserendo dopo la lettera della nota un segno più (+) o un segno di numero (#) per i diesis e un segno meno (-) per i bemolle. Questi segni non possono essere utilizzati con una nota indicata dal comando N e un numero, dato che ogni numero corrisponde a un tono diverso. Ad esempio, il comando N38 indica la nota C# nella terza ottava, come riportato nella Tabella 12.2.

Il comando L seguito da un numero permette di specificare la lunghezza di una nota; ad esempio L2 N0 suona una pausa di 1 mezzo. La durata viene indicata come parte frazionaria di un nota intera: L1 specifica una nota intera (4 battiti), L2 un mezzo

Tabella 12.2 Le note e i numeri corrispondenti

Ottava	Nota	Numero	Ottava	Nota	Numero
0	C	1	3	F#/G♭	43
0	C#/D♭	2	3	G	44
0	D	3	3	G#/A♭	45
0	D#/E♭	4	3	A	46
0	E	5	3	A#/B♭	47
0	F	6	3	B	48
0	F#/G♭	7	4	C	49
0	G	8	4	C#/D♭	50
0	G#/A♭	9	4	D	51
0	A	10	4	D#/E♭	52
0	A#/B♭	11	4	E	53
0	B	12	4	F	54
1	C	13	4	F#/G♭	55
1	C#/D♭	14	4	G	56
1	D	15	4	G#/A♭	57
1	D#/E♭	16	4	A	58
1	E	17	4	A#/B♭	59
1	F	18	4	B	60
1	F#/G♭	19	5	C	61
1	G	20	5	C#/D♭	62
1	G#/A♭	21	5	D	63
1	A	22	5	D#/E♭	64
1	A#/B♭	23	5	E	65
1	B	24	5	F	66
2	C	25	5	F#/G♭	67
2	C#/D♭	26	5	G	68
2	D	27	5	G#/A♭	69
2	D#/E♭	28	5	A	70
2	E	29	5	A#/B♭	71
2	F	30	5	B	72
2	F#/G♭	31	6	C	73
2	G	32	6	C#/D♭	74
2	G#/A♭	33	6	D	75
2	A	34	6	D#/E♭	76
2	A#/B♭	35	6	E	77
2	B	36	6	F	78
3	C	37	6	F#/G♭	79
3	C#/D♭	38	6	G	80
3	D	39	6	G#/A♭	81
3	D#/E♭	40	6	A	82
3	E	41	6	A#/B♭	83
3	F	42	6	B	84

(2 battiti), L4 un quarto (1 battito) e così via fino a L64, un sessantaquattresimo (1/8 di un battito), e imposta la lunghezza delle note successive, fino a quando non viene fornito un nuovo comando L. Oppure è possibile specificare la lunghezza di una nota particolare (senza influenzare le note successive), inserendo soltanto il numero della lunghezza dopo la nota; ad esempio C8 suona un C lungo un ottavo.

È anche possibile prolungare la durata di una nota per metà della sua lunghezza base, inserendo dopo la lunghezza un punto. Ad esempio, L1.D specifica una nota intera D che viene tenuta per un'ulteriore mezza nota, per una durata totale di 1 e 1/2 (6 battiti). Questa notazione può anche essere utilizzata per trascrivere qualsiasi nota con punto in uno spartito.

Dato che in molte canzoni si utilizzano diverse ottave, QBasic offre un metodo vantaggioso per abbassare o alzare le note: il segno di maggiore (>) alza la nota successiva di un'ottava, mentre il segno di minore (<) la abbassa di un'ottava. Questo valore rimane valido fino a quando non ne viene specificato un altro, oppure finché non si utilizza il comando O per indicare un'ottava particolare. Nella canzone "Red River Valley", le note sono nelle ottave 2 e 3.

Il tempo e altri valori

QBasic consente di controllare la velocità (il tempo) della musica che si compone. Il comando T nell'istruzione PLAY indica il tempo in termini di quante note di un quarto vengono suonate in un minuto, in un intervallo da 32 a 255; il valore di default è 120. Si noti che utilizzando un tempo veloce (ad esempio T200), le noti brevi potrebbero non suonare, dato che l'intervallo è più breve rispetto al ciclo dell'orologio di sistema che QBasic utilizza per tenere conto del tempo.

La lunghezza delle note può anche essere modificata utilizzando i seguenti comandi:

ML (legato)	La lunghezza effettiva della nota è intera
MN (normale)	La lunghezza effettiva della nota è 7/8.
MS (staccato)	La lunghezza effettiva della nota è 3/4.

Questi comandi consentono di creare quegli effetti che corrispondono allo spirito della musica.

Esecuzione dei comandi da una variabile stringa

È possibile assegnare una serie di comandi PLAY a una variabile stringa e quindi eseguire il suo contenuto, inserendola in un'istruzione PLAY. Ad esempio, le istruzioni di seguito riportate consentono di suonare una scala musicale.

```
scala$ = "CDEFGAB"
PLAY "X" + VARPTR$(scala$)
```

Per utilizzare una variabile stringa in un'istruzione PLAY, è necessario innanzitutto inserire una X nella stringa di comandi, seguita da una serie di doppi apici di chiusura, aggiungendo infine una chiamata alla funzione VARPTR$ (questa funzione restituisce

l'indirizzo della variabile stringa, in modo che PLAY possa individuare i comandi contenuti in essa).

LA MUSICA IN SOTTOFONDO

Il comando MB di PLAY indica che la musica viene suonata in sottofondo, mentre il programma continua l'esecuzione.

Quando si utilizza questa caratteristica, le note generate dall'istruzione PLAY non vengono suonate immediatamente, ma vengono memorizzate in un buffer interno. Dopo che il programma ha eseguito un'istruzione, verifica il buffer musicale per controllare se vi sono ancora note da suonare e in caso positivo ne suona alcune. Il numero di note che viene suonato dipende dalla durata delle stesse e dal fatto che sia trascorso del tempo da quando è stata suonata l'ultima.

Dato che il buffer può contenere soltanto 32 note, se si desidera suonare una melodia più lunga è necessario eseguire più di un'istruzione PLAY per riempire nuovamente il buffer.

Se si desidera suonare continuamente una melodia in sottofondo, meglio impostare una gestione degli eventi che consenta di riempire nuovamente il buffer musicale ogniqualvolta il numero di note scende al di sotto di un determinato valore. A questo scopo è necessario innanzitutto creare una subroutine contenente le istruzioni PLAY che riempiono il buffer (per assicurarsi che la musica venga suonata in sottofondo è sufficiente inserire nel programma un'istruzione PLAY che contiene il comando MB). Poi, occorre indicare quella subroutine in un'istruzione ON PLAY GOSUB. Ad esempio, l'istruzione:

```
ON PLAY(5) GOSUB AltraMusica
```

indica che quando rimangono meno di cinque note nel buffer musicale il programma deve eseguire la subroutine AltraMusica. Probabilmente è necessario sperimentare il numero di note indicato; se troppo basso, il buffer potrebbe svuotarsi prima che QBasic lo controlli, se invece il numero è troppo alto, il programma potrebbe riempire nuovamente il buffer con una frequenza tale da rallentare l'esecuzione del programma.

Dopo aver impostato la subroutine musicale con l'istruzione ON PLAY GOSUB, occorre utilizzare PLAY ON per attivare il meccanismo di gestione degli eventi e quindi riempire inizialmente il buffer, richiamando la subroutine con un'istruzione GOSUB. La struttura generale del programma è simile al seguente listato:

```
ON PLAY(5) GOSUB SubMusic      'imposta la subroutine musicale
PLAY ON      'attiva la gestione
PLAY "MB"    'indica musica in sottofondo
GOSUB SubMusic     'inizia la musica
'altre istruzioni vengono eseguite durante la musica
'il buffer viene regolarmente riempito da SubMusic
END

SubMusic
   'PLAY istruzioni
   RETURN
```

QBasic fornisce altre due istruzioni che possono essere utilizzate con PLAY: la prima è PLAY OFF, che interrompe la gestione degli eventi impedendo che il programma esegua la subroutine musicale, in modo che la musica in sottofondo si interrompa, dopo che sono state suonate tutte le note contenute nel buffer; anche la seconda istruzione, PLAY STOP, impedisce l'esecuzione della subroutine musicale, anche se in questo caso il programma "ricorda" se il numero di note nel buffer musicale è sceso al di sotto della soglia specificata e, quando viene eseguita la successiva istruzione PLAY ON, la subroutine musicale esegue e riempie il buffer. PLAY STOP può essere utilizzata per disattivare temporaneamente la subroutine musicale (in modo che la parte principale del programma non venga rallentata) e PLAY ON per ripristinare la musica in sottofondo.

Se si preferisce eseguire un'operazione soltanto quando il buffer musicale contiene un determinato numero di note, è sufficiente utilizzare la funzione PLAY per verificare il numero di note restanti nel buffer.

ALTRE ISTRUZIONI RELATIVE AL SUONO

L'istruzione BEEP fa emettere un suono all'altoparlante interno del computer, proprio come se fosse stata eseguita l'istruzione PRINT CHR$(7).

L'istruzione SOUND produce un suono di una frequenza e una durata specificate e può essere utilizzata per creare diversi effetti sonori (il programma di esempio riportato nella voce SOUND crea il suono della sirena di una macchina della polizia). La parola chiave SOUND deve essere seguita dalla frequenza in cicli per secondo, o hertz (un valore nell'intervallo da 37 a 32767), da una virgola e dalla durata in cicli di clock (l'orologio del computer esegue un ciclo 18.2 volte per secondo). Quindi, l'istruzione:

```
SOUND 440, 18.2
```

genera una nota di 440 hertz (la nota A) per 1 secondo. L'istruzione SOUND è più versatile di PLAY, dato che non è limitata alle note musicali riconosciute, ma d'altro canto i suono prodotti da questa istruzione non possono essere suonati in sottofondo.

Bibliografia

Enders, Bernd, and Bob Peterson. The Waite Group. *BASIC Primer for the IBM PC & XT.* New York, N.Y.: New American Library, 1984.

BEEP

Si consulti anche

PLAY (Istruzione), SOUND

Scopo

L'istruzione BEEP fa in modo che l'altoparlante di sistema emetta un suono acustico. Questo tipo di suono viene spesso utilizzato per richiamare l'attenzione su un errore o per avvertire l'utente delle eventuali conseguenze delle azioni che sta compiendo, ad esempio la cancellazione di un file.

Sintassi

```
BEEP
```

Utilizzo

```
BEEP
```

Fa emettere un suono acustico all'altoparlante di sistema.

Descrizione

L'istruzione BEEP fa emettere un suono acustico della durata di circa mezzo secondo all'altoparlante di sistema. Lo stesso risultato può essere ottenuto con l'istruzione PRINT CHR$(7).

Non è possibile controllare la durata di un singolo suono acustico, ma è possibile farlo sembrare più lungo ripetendo l'istruzione BEEP in un ciclo. Dato che non è possibile controllare il tono del suono, se si desidera creare un suono diverso o una nota musicale è necessario utilizzare l'istruzione SOUND o PLAY.

Esempio

Il programma di seguito riportato utilizza un ciclo per richiedere l'immissione di uno dei seguenti caratteri: A, B, C o D (in lettere maiuscole o minuscole). Se l'utente immette un carattere non valido, il programma emette un suono acustico richiamando BEEP e richiedendo di provare di nuovo.

```
CLS
DO
  PRINT "Immettere A, B, C, o D."
  INPUT carattere$
  scelta$ = UCASE$(carattere$)
  IF INSTR("ABCD", scelta$) THEN
      PRINT "Molto bene!"
      EXIT DO
  ELSE
      BEEP
      PRINT "Per favore, ritenta."
  END IF
LOOP
```

ON PLAY GOSUB

Si consulti anche

PLAY (Funzione), PLAY (Istruzione), PLAY ON/OFF/STOP

Scopo

L'istruzione ON PLAY GOSUB indica l'inizio di una subroutine che viene eseguita ogniqualvolta il numero di note rimaste nel buffer musicale scende al di sotto del livello specificato; può essere utilizzata per indicare la subroutine che inserisce nel buffer musicale le note da suonare in sottofondo, mentre il programma continua l'esecuzione.

Sintassi

```
ON PLAY(note) GOSUB {etichetta | riga}
```

note è il numero che indica il numero minimo di note che possono trovarsi nel buffer musicale, prima che il programma esegua la subroutine. Questo valore deve essere un numero intero nell'intervallo da 1 a 32.

etichetta o riga è l'etichetta o il numero di riga della subroutine da eseguire, quando nel buffer rimane un numero di note inferiore rispetto a quello specificato dal parametro note.

Utilizzo

```
ON PLAY(3) GOSUB AltraMusica
```

Indica che viene eseguita la subroutine AltraMusica, quando nel buffer musicale sono rimaste meno di tre note da suonare.

Descrizione

L'istruzione ON PLAY GOSUB imposta una gestione degli eventi relativa al suono, che consente di riempire il buffer musicale. Questa istruzione indica la subroutine a cui passa il programma, quando il numero di note memorizzate nel buffer diventa inferiore al valore specificato. Questa procedura permette di mantenere pieno il buffer, in modo che la musica possa essere suonata in sottofondo , fino a quando la gestione degli eventi non viene disattivata o sospesa. Dopo aver richiamato l'istruzione ON PLAY GOSUB, è necessario chiamare anche PLAY ON per iniziare la procedura di gestione degli eventi.

Errori

Se viene specificato un numero di note accodate che non rientra nell'intervallo da 1 a 32, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita".

Esempio

Per una dimostrazione dell'utilizzo dell'istruzione ON PLAY GOSUB, si consulti il programma di esempio riportato alla voce PLAY ON/OFF/STOP.

PLAY (Funzione)

Si consulti anche

ON PLAY GOSUB, PLAY (Istruzione), PLAY ON/OFF/STOP

Scopo

La funzione PLAY restituisce il numero di note che devono essere ancora suonate nel buffer musicale; può essere utilizzata per verificare lo stato del buffer; nel caso fosse quasi vuoto, è possibile eseguire un'istruzione PLAY per riempirlo.

Sintassi

```
PLAY(n)
```

n è un valore fittizio, utilizzato per distinguere la funzione PLAY dall'omonima istruzione, e può avere qualsiasi valore numerico.

Utilizzo

```
IF PLAY(1) < 10 THEN PLAY "C D E F G A B"
```

Inserisce le note di una scala nel buffer musicale, se sono rimaste meno di 10 note da suonare.

Descrizione

La funzione PLAY restituisce il numero di note restanti nel buffer musicale. Il buffer contiene delle note soltanto quando la musica viene suonata in sottofondo . Facendo in modo che il programma verifichi questo valore periodicamente, è possibile eseguire istruzioni PLAY per riempire nuovamente il buffer, in modo da continuare a suonare musica in sottofondo.

Se la musica viene eseguita in primo piano (il valore di default dell'istruzione PLAY, quando non viene specificato il comando MB), la funzione PLAY restituisce 0.

Commenti

Se il programma è lungo o presenta una variazione considerevole nel proprio percorso di esecuzione, la funzione PLAY potrebbe non essere eseguita con una frequenza tale da impedire che il buffer musicale si svuoti (provocando in questo modo un'interruzione della musica). In questi casi, è meglio utilizzare l'istruzione ON PLAY GOSUB, che consente al programma di passare a una subroutine che riempie nuovamente il buffer, quando quest'ultimo è quasi vuoto.

Esempio

Il programma di seguito riportato suona ripetutamente una scala in sottofondo , richiamando ogni volta la funzione PLAY mediante un ciclo e indicando il numero di note presenti nel buffer musicale. Quando quest'ultimo contiene meno di tre note, il programma lo riempie nuovamente, richiamando l'istruzione PLAY.

```
PLAY "MB"                           'imposta il modo musicale in sottofondo
CLS


DO
  LOCATE 5, 1
  PRINT "Numero delle note rimaste da suonare:";
  IF PLAY(1) < 3 THEN 'riempie il buffer musicale se rimangono meno di 3 note
```

```
      PLAY "< C D E F G A B > C"
    END IF
    PRINT PLAY(1)
LOOP UNTIL INKEY$ <> ""          'cicla fino a quando l'utente preme un tasto
END
```

PLAY (Istruzione)

Si consulti anche

ON PLAY GOSUB, PLAY ON/OFF/STOP

Scopo

L'istruzione PLAY suona il numero specificato di note musicali. È possibile controllare il tono e la durata di ogni nota e inserire alcune pause. La musica può essere suonata in primo piano o in sottofondo.

Sintassi

```
PLAY stringa
```

stringa è un letterale stringa, una variabile stringa o un'espressione stringa che contiene una serie di comandi musicali. Questi comandi possono essere immessi in lettere maiuscole o minuscole e separati da spazi o punti e virgola per rendere la stringa più leggibile.

Utilizzo

```
PLAY "C D E F G A B"
```

Suona una scala.

```
PLAY "X" + VARPTR$(cmd$)
```

Suona i comandi presenti nella variabile stringa cmd$.

Descrizione

L'istruzione PLAY utilizza un linguaggio macro che consente di suonare delle melodie. La stringa tra doppi apici (oppure la variabile stringa) che segue la parola chiave PLAY comprende una serie di comandi macro che indicano il tono e la durata delle note, il tempo e se la musica deve essere suonata in sottofondo o in primo piano.

Tono

Le note possono essere suonate in un massimo di sette ottave. Il comando O deve essere utilizzato con un numero compreso nell'intervallo da 0 a 6 per indicare l'ottava delle note successive (la quarta ottava, o O3, contiene il C centrale). É anche possibile utilizzare un segno di maggiore (>) per specificare un'ottava più alta o il segno di minore (<) per indicarne una più bassa (le specifiche sopra l'ottava 6 o sotto l'ottava 0 vengono ignorate).

Per indicare il tono da suonare è possibile utilizzare il comando N seguito dal numero della nota (da 0 a 84), oppure semplicemente la lettera musicale (da A a G),

tenendo presente che, quando si utilizza quest'ultimo metodo, QBasic suona quella nota particolare all'interno dell'ultima ottava specificata. Il comando N0 indica una pausa. Nella notazione letterale, un segno più (+) o il segno del numero (#) indicano un diesis, mentre il segno meno (-) indica un bemolle. I diesis e i bemolle influiscono soltanto sulla nota corrente.

È anche possibile specificare una pausa, utilizzando il comando P seguito dalla frazione della nota intera (nell'intervallo da 1 a 64), che indica la lunghezza.

Durata

Per specificare il valore della durata, deve essere utilizzato il comando L seguito dai seguenti valori: 1 è una nota intera, 2 due quarti, 4 un quarto, 8 un ottavo, 16 un sedicesimo e 32 un trentaduesimo. Il valore della nota più breve è 64 (un sessantaquattresimo). Dopo aver specificato la lunghezza con il comando L, tutte le note successive utilizzano il valore indicato, fino a quando non viene modificato. Se si desidera cambiare la lunghezza soltanto di una nota, è sufficiente specificare un numero dopo di essa; ad esempio, il comando PLAY "L4 C G A" suona C, G e A come note di un quarto. Se il comando successivo fosse PLAY "F8 B C", F sarebbe una nota di un ottavo, mentre B e C verrebbero suonate con durata di un quarto (l'ultima lunghezza specificata).

In base alla notazione musicale, è anche possibile far seguire una nota da un punto per incrementare di metà la lunghezza effettiva. Ad esempio, il comando PLAY "L8 G A." suona G come una nota di un ottavo e A come nota di un ottavo e mezzo ("un ottavo col punto").

La lunghezza di tutte le note può anche essere modificata dai seguenti comandi:

ML (legato)	La lunghezza effettiva della nota è intera.
MN (normale)	La lunghezza effettiva della nota è 7/8.
MS (staccato)	La lunghezza effettiva della nota è 3/4.

Ognuno di questi comandi ha effetto fino a quando non ne viene specificato un altro.

La lunghezza di una nota è la durata relativa della stessa, ovvero per quanto tempo una nota suona in base al valore della nota intera. La quantità di tempo (in secondi) è determinata dal *tempo*, cioè dal ritmo in base al quale viene suonata la musica. In PLAY, il tempo viene espresso in note di un quarto per minuto e viene specificato dal comando T seguito da un numero, nell'intervallo da 32 a 255. Il valore di default è T120, che significa 120 note di un quarto suonate al minuto.

Si noti che il ritmo di interrupt dell'orologio di sistema limita la brevità e la rapidità delle note. Ad esempio, se il tempo viene impostato a 255 il programma non è in grado di suonare le note di un sessantaquattresimo (T255 L64), dato che il sistema non è in grado di verificare l'orologio per cronometrare e produrre la nota in un intervallo tanto breve.

Musica in primo piano e in sottofondo

PLAY comprende due comandi che indicano la modalità in base alla quale il programma suona le note specificate. Per default, la musica suona in primo piano

(indicato dal comando MF); questo significa che QBasic esegue le istruzioni PLAY e suona le note specificate, senza eseguire alcuna altra istruzione fino a quando la musica non è terminata.

Se invece viene utilizzato il comando MB (musica in sottofondo), le note indicate nell'istruzione PLAY vengono inserite in un buffer e suonate in sottofondo, mentre l'esecuzione del programma continua. Dopo che sono state eseguite tutte le istruzioni, QBasic verifica il buffer e suona ancora alcune note. Il numero di note suonate viene determinato dalla durata delle note e dal fatto che alcune stiano ancora suonando, infatti QBasic cerca di mantenere il tempo impostato. Dato che il buffer può contenere un massimo di 32 note, se si desidera che la musica prosegua senza interruzione è necessario impedire che si svuoti.

Utilizzo di una variabile stringa

Per comodità, è possibile assegnare una serie di comandi PLAY a una variabile stringa e quindi utilizzarla in un'istruzione PLAY. Ad esempio, l'istruzione cmd$ = "C D E" assegna tre comandi PLAY a questa variabile; quando quest'ultima viene inserita come sottostringa all'interno della stringa di comandi PLAY, QBasic esegue i comandi presenti nella sottostringa. Per utilizzarla, è sufficiente immettere una X al termine del comando PLAY e quindi aggiungere una chiamata alla funzione VARPTR$, come nella seguente istruzione:

```
PLAY "O3 A B- X" + VARPTR$(cmd$)
```

Se si desidera utilizzare una variabile di tipo intero per specificare il valore che segue il comando, è necessario che, dopo la lettera che designa il comando, venga inserito un segno di uguale e una serie di doppi apici. Quindi, la variabile deve essere aggiunta in una chiamata a VARPTR$ alla stringa di comandi. Ad esempio, se la variabile ottava% contiene il numero dell'ottava di una particolare melodia, è possibile impostare l'ottava con la seguente istruzione:

```
PLAY "O=" + VARPTR$(ottava%)
```

Errori

Se viene specificato un intero non compreso nell'intervallo di un determinato comando PLAY, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita". QBasic visualizza lo stesso messaggio se nella stringa di comando viene utilizzata una lettera che non è un comando.

Esempio

Il programma di seguito riportato utilizza le istruzioni PLAY per suonare la canzone dei Beatles "Here Comes the Sun".

```
PLAY "T255"                'imposta il tempo a 255 note di un quarto per minuto
PLAY "O3"
PLAY "< < L2 A > > > L4 C# < A B > L2 C#."
PLAY "L2 NO L4 C# < L2 B A F#"
PLAY "A B A F# L1 E."
PLAY "< < L2 A > > > L4 C# < A B > L2 C#."
PLAY "L2 NO L4 C# < L2 B A L2 F#."
PLAY "> L4 C# < L2 B A L1 B B L1 NO"
```

```
PLAY "< < L2 A > > > L4 C# < L2 B > L2 C# < L2 A."
PLAY "> L4 C# < A B > L2 C#."
PLAY "< < L2 A > > > L4 C# < L2 B > C# < L2 A."
PLAY "L4 NO A L2 B A"
PLAY "L2 NO > L2 C# < B A NO"
PLAY "L4 F# A B E A B D A B A G# F# E"
```

PLAY ON/OFF/STOP

Si consulti anche

PLAY (Funzione), PLAY (Istruzione)

Scopo

Le istruzioni PLAY attivano, disattivano o sospendono la gestione degli eventi di suono, consentendo di riempire nuovamente il buffer musicale in modo continuo, mentre la musica viene suonata in sottofondo e il programma esegue altre istruzioni.

Sintassi

```
PLAY ON
PLAY OFF
PLAY STOP
```

Utilizzo

```
ON PLAY(3) GOSUB Riempi
PLAY ON
```

Riempie nuovamente il buffer musicale, come indicato nell'istruzione ON PLAY GOSUB.

```
PLAY OFF
```

Sospende la gestione degli eventi di suono e riempie nuovamente il buffer musicale.

Descrizione

Quando viene suonata della musica in sottofondo (inserendo il comando MB in un'istruzione PLAY), le note vengono memorizzate in un buffer interno, anziché suonate immediatamente. QBasic verifica il buffer dopo l'esecuzione di ogni istruzione successiva e suona il maggior numero di note possibili. Quando il numero di note restanti scende al di sotto del numero indicato nell'istruzione ON PLAY GOSUB, viene eseguita la subroutine indicata in quell'istruzione; generalmente questa subroutine esegue istruzioni PLAY oppure richiama altre subroutine che eseguono le subroutine PLAY, riempiendo in questo modo nuovamente il buffer.

Per disattivare completamente la musica in sottofondo è sufficiente utilizzare l'istruzione PLAY OFF, che impedisce l'esecuzione della subroutine ON PLAY GOSUB. In questo modo, il buffer rimane senza note e la musica in sottofondo termina.

Per sospendere temporaneamente questa musica (ad esempio, quando il programma sta eseguendo istruzioni che potrebbero essere influenzate dalle ripetute interruzioni che consentono di riempire il buffer), è sufficiente utilizzare l'istruzione PLAY STOP.

Le noti restanti nel buffer vengono suonate e quindi la musica di sottofondo si interrompe. Per riprenderla, è sufficiente eseguire un'altra istruzione PLAY ON. Se il numero di note è inferiore a quello indicato nell'istruzione ON PLAY GOSUB, dopo che è stata richiamato PLAY STOP, QBasic imposta un flag interno che indica che il buffer deve essere nuovamente riempito, in modo che il programma esegua la subroutine quando viene richiamata PLAY ON.

Esempio

Il programma di seguito riportato utilizza l'istruzione PLAY ON per attivare la gestione degli eventi di suono e sfrutta l'istruzione ON PLAY GOSUB per inviare il controllo alla subroutine Ripeti, quando il buffer deve essere riempito. Si noti che l'istruzione "PLAY MB T255" indica che la musica deve essere suonata in sottofondo e imposta un tempo. La canzone che viene suonata è "Here Comes the Sun" dei Beatles.

Il ciclo FOR principale traccia progressivamente ampi cerchi per modificare i colori che rappresentano il sole, mentre il ciclo DO esegue il programma fino a quando l'utente non preme ESC.

Le istruzioni IF controllano il flag variabile musica. La prima volta che viene premuto un tasto (che non sia ESC), musica viene impostato a 0 e viene eseguita l'istruzione PLAY STOP. Questo permette di sospendere la gestione degli eventi e di interrompere la musica. Quando un utente preme nuovamente un tasto, musica viene impostato a 1 e l'istruzione PLAY ON riempie nuovamente il buffer musicale e la musica riprende. Se l'istruzione PLAY STOP viene sostituita con PLAY OFF, la musica non riprende.

La subroutine Ripeti contiene le istruzioni PLAY che riempiono il buffer musicale.

```
SCREEN 9                'per EGA; usa 1 per CGA, 12 per VGA
maxX% = 640             'per EGA; usa 320 per CGA
maxY% = 350             'per EGA; usa 200 per CGA, 480 per VGA

x = maxX% / 2
y = maxY% / 2           'posizione iniziale per il cerchio
maxColori% = 9
inizioRag = maxY% / 16  'inizio del raggio del cerchio

DIM colori(maxColori%)

FOR cl% = 1 TO 9        'legge i valori del colore di primo piano
   READ colori(cl%)
NEXT cl%

'imposta il suono
ON PLAY(3) GOSUB Ripeti  'riempie il buffer quando rimangono tre note o meno
GOSUB Ripeti             'riempie inizialmente il buffer musicale
PLAY ON                  'attiva la gestione degli eventi musicali
musica = -1              'flag che indica se la musica sta suonando

DO                       ' ciclo principale che mostra il grafico
   FOR cl% = 1 TO maxColori%         'ciclo FOR che attraversa i valori
       coloreCerchio = colori(cl%)
       raggio = inizioRag * cl%
       CIRCLE (x, y), raggio, (coloreCerchio)     'disegna il cerchio
       PAINT (x, y), coloreCerchio                'riempie il cerchio
   NEXT cl%
   tasto$ = INKEY$
   IF tasto$ = CHR$(27) THEN EXIT DO   'esce dal ciclo se l'utente preme Esc
   FOR n = 1 TO 500: NEXT n
```

```
    PAINT (x, y), 0                     'cancella il cerchio
    IF tasto$ <> "" THEN                'modifica lo status di PLAY
       IF musica THEN
          PLAY STOP                     'mette in pausa la gestione dell'evento
          musica = 0
       ELSE
          PLAY ON                       'abilita la gestione dell'evento
          musica = 1
       END IF
    END IF
LOOP UNTIL tasto$ = CHR$(27)            'esce dal ciclo se l'utente preme Esc
PLAY OFF
END

'subroutine per riempire il buffer musicale
Ripeti:
    PLAY "MB T255"
    PLAY "O3"
    PLAY "< < L2 A > > > L4 C# < A B > L2 C#."
    PLAY "L2 NO L4 C# < L2 B A F#"
    PLAY "A B A F# L1 E."
    PLAY "< < L2 A > > > L4 C# < A B > L2 C#."
    PLAY "L2 NO L4 C# < L2 B A L2 F#."
    PLAY "> L4 C# < L2 B A L1 B B L1 NO"

    PLAY "< < L2 A > > > L4 C# < L2 B > L2 C# < L2 A."
    PLAY "> L4 C# < A B > L2 C#."
    PLAY "< < L2 A > > > L4 C# < L2 B > C# < L2 A."
    PLAY "L4 NO A L2 B A"
    PLAY "L2 NO > L2 C# < B A NO"
    PLAY "L4 F# A B E A B D A B A G# F# E"
    RETURN

'colori di primo piano del cerchio
DATA 4,12,5,13,5,12,14,12,14
```

SOUND

Si consulti anche

BEEP, PLAY (Istruzione)

Scopo

L'istruzione SOUND fa emettere all'altoparlante del computer un suono di una specificata durata e frequenza; può essere utilizzata per creare effetti sonori diversi rispetto alle note musicali standard generate dall'istruzione PLAY.

Sintassi

```
SOUND frequenza, durata
```

frequenza è un'espressione di tipo intero nell'intervallo da 37 a 32767 e rappresenta la frequenza del suono in cicli per secondo (hertz).

durata è un'espressione di tipo intero nell'intervallo da 0 a 65535 e rappresenta la lunghezza del suono in cicli di clock.

Utilizzo

```
SOUND 440, 18.2
```

Emette un suono di 440-hertz (A musicale) per 1 secondo (18,2 cicli di clock).

```
SOUND 500, 0
```

Termina il suono attuale. Qualsiasi durata che abbia valore 0 interrompe il suono attualmente in esecuzione.

Descrizione

L'istruzione SOUND genera un suono con frequenza uguale al numero specificato di cicli per secondo (hertz) e di lunghezza uguale al numero indicato di cicli di clock. Dato che l'orecchio umano è in grado di sentire i suoni con frequenza compresa tra 20 e 20000 hertz, è necessario mantenersi in questo intervallo. Una durata di 0 cicli in un'istruzione SOUND interrompe qualsiasi suono in fase di esecuzione.

L'altoparlante del computer non è stato progettato per riprodurre suoni più alti di alcune migliaia di hertz. Al contrario di PLAY, SOUND non può essere eseguita in sottofondo , tuttavia il suono continua fino al termine previsto, mentre QBasic esegue le istruzioni successive.

Errori

Se con SOUND viene utilizzato un valore stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Qualsiasi valore numerico al di fuori dell'intervallo dà come risultato il messaggio: "Chiamata di funzione non lecita".

Esempio

Il programma di seguito riportato suona un glissando ascendente e discendente che produce un effetto sonoro analogo al suono di una sirena della polizia. Per crearlo, il programma richiama l'istruzione SOUND in un ciclo FOR, che modifica, per 4 hertz a ogni iterazione, la nota suonata.

```
CLS
PRINT "                  VIA!"
PRINT "<Premi un tasto qualsiasi per terminare>"
DO
  FOR i% = 440 TO 880 STEP 4
      SOUND i%, i% / 880
      IF INKEY$ <> "" THEN EXIT DO
  NEXT
  FOR i% = 880 TO 440 STEP -4
      SOUND i%, i% / 880
      IF INKEY$ <> "" THEN EXIT DO
  NEXT
LOOP UNTIL INKEY$ <> ""
```

Capitolo 13

La penna ottica e il joystick

INTRODUZIONE

In questo capitolo vengono esaminati due dispositivi ausiliari di input collegabili ai personal computer, ovvero la penna ottica e il joystick. In modo analogo al mouse, al trackball e al touch pad, la penna ottica e il joystick sono dispositivi di puntamento.

Dato che i concetti fondamentali utilizzati per programmare la penna ottica sono virtualmente identici a quelli sfruttati per il joystick, in questa parte le caratteristiche specifiche dei due dispositivi vengono affrontate separatamente, mentre la programmazione viene riunita. Nella Tabella 13.1 sono riportate in ordine alfabetico le istruzioni e le funzioni relative alla penna ottica e al joystick.

PENNE OTTICHE

La penna ottica è un puntatore a forma di penna, che viene utilizzato per interagire direttamente con lo schermo. Ne esistono di due tipi: per attivare il tipo *push-tip* (a contatto diretto), è sufficiente premere con la punta sopra un'area appropriata dello schermo (ad esempio un menu), mentre per attivare il tipo *touch-ring* (a pulsante), è necessario tenerlo leggermente sopra l'area dello schermo appropriata, premendo l'apposito pulsante.

Le penne ottiche vengono utilizzate principalmente nelle operazioni effettuate nei punti vendita, nella progettazione CAD e talvolta nel software educativo: per effettuare la scelta desiderata, è sufficiente puntare ciò che si desidera e attivare la penna. Al contrario del mouse, la penna ottica non richiede al computer di convertire un movimento meccanico (sul tavolo) in un movimento del cursore (sullo schermo). Sebbene l'utilizzo della penna possa risultare più intuitivo rispetto a quello del mouse, con essa è necessario raggiungere lo schermo per immettere un comando. Senza dimenticare che il tipo push-tip rischia di graffiare o sporcare la superficie dello schermo.

Tabella 13.1 Le istruzioni e le funzioni relative alla penna ottica e al joystick

Istruzione o funzione	Descrizione
ON PEN GOSUB	Indica la subroutine da eseguire quando la penna ottica è attiva ed è abilitata la gestione degli eventi.
ON STRIG GOSUB	Indica la subroutine da eseguire quando viene premuto un determinato pulsante del joystick ed è attiva la gestione degli eventi.
PEN	Restituisce la posizione x o y (colonna o riga) della penna ottica.
PEN ON/OFF/STOP	Attiva, disattiva o sospende la gestione degli eventi relativa alla penna ottica.
STICK	Restituisce la posizione x o y (colonna o riga) del joystick.
STRIG	Indica se un determinato pulsante del joystick è stato premuto.
STRIG ON/OFF/STOP	Attiva, disattiva o sospende la gestione degli eventi relativa al joystick.

JOYSTICK

I joystick vengono utilizzati con maggiore frequenza rispetto alle penne ottiche; qualche volta sono impiegati nelle applicazioni educative, ma sono sfruttati soprattutto nei giochi e nelle simulazioni. Oltre alle simulazioni di volo e ai giochi arcade, i joystick possono essere utilizzati anche per spostare i pezzi nei giochi di strategia. Questi dispositivi variano nel numero di pulsanti che presentano e nella struttura. Probabilmente è necessario effettuare alcune prove per determinare in che modo i pulsanti del joystick utilizzato corrispondono ai pulsanti "inferiore" e "superiore" riconosciuti da QBasic.

IMPOSTAZIONE DELLA GESTIONE DEGLI EVENTI

Per utilizzare la penna ottica o il joystick nei propri programmi, è necessario acquisire dimestichezza con il concetto di gestione degli eventi. Un programma esegue una gestione degli eventi verificando l'input della penna ottica o del joystick ed eseguendo quindi la subroutine opportuna.

Innanzitutto, occorre impostare la gestione degli eventi specificando il nome delle subroutine a cui deve passare l'esecuzione, quando viene attivato uno di questi due dispositivi. Per impostare la gestione degli eventi relativi alla penna ottica è sufficiente utilizzare l'istruzione ON PEN GOSUB, mentre per il joystick l'istruzione corretta è ON STRIG GOSUB. Oltre a indicare la subroutine, in quest'ultima istruzione è necessario inserire un numero che specifichi il joystick (è possibile collegarne due al computer) e il pulsante a cui deve rispondere la gestione degli eventi. Nella Tabella 13.2 sono riportati i numeri e i corrispondenti pulsanti.

Tabella 13.2 I numeri utilizzati con ON STRING GOSUB e i pulsanti corrispondenti

Numero	Pulsante
1	Pulsante inferiore del primo joystick
2	Pulsante inferiore del secondo joystick
3	Pulsante superiore del secondo joystick
4	Pulsante superiore del secondo joystick

A questo punto occorre attivare la gestione degli eventi, richiamando PEN ON per le penne ottiche e STRIG ON per i joystick. É necessario indicare il pulsante del joystick che verrà gestito, inserendo nella relativa istruzione il numero corretto riportato nella Tabella 13.2. Si noti che, anche se non si utilizza la gestione degli eventi per la penna ottica, è sempre necessario inserire l'istruzione PEN ON, prima di richiamare qualsiasi istruzione (tranne ON PEN GOSUB) che operi con questo dispositivo.

Il listato di seguito riportato delinea un programma che utilizza la penna ottica.

```
ON PEN GOSUB Scegli       'specifica la subroutine per la gestione della penna
                          'ottica
PEN ON                    'abilita la gestione della penna ottica
.
.
.
END      'interrompe l'esecuzione prima di arrivare alla subroutine

Scegli:
     'istruzioni eseguite quando l'utente attiva la penna ottica
     .
     .
     .
     RETURN
```

Il listato seguente invece delinea un programma che utilizza il joystick.

```
ON STRIG(0) GOSUB Scegli1    'subroutine per il pulsante inferiore, joystick 1
ON STRIG(4) GOSUB Scegli2    'subroutine per il pulsante superiore, joystick 1
STRIG(0) ON                  'abilita la gestione del pulsante inferiore
STRIG(4) ON                  'abilita la gestione del pulsante superiore
.
.
.
END      'interrompe l'esecuzione prima di arrivare alla subroutine

Scegli1:
     'istruzioni eseguite quando l'utente preme il pulsante inferiore
     .
     .
     .
     RETURN

Scegli2:
     'istruzioni eseguite quando l'utente preme il pulsante superiore
     .
     .
     .
     RETURN
```

Dato che la configurazione dei pulsanti varia in base al joystick prescelto, un programma che sfrutta questo strumento deve richiedere all'utente di premere ogni pulsante del joystick per definire quale sia il pulsante inferiore e quello superiore. Se il programma utilizza due joystick e si desidera che sia in grado di gestire entrambi, è sufficiente intercettare le azioni di ambedue utilizzando una subroutine per entrambi i pulsanti inferiori e un'altra per quelli superiori.

Se si desidera disattivare la gestione degli eventi relativa alla penna ottica o al joystick, è sufficiente utilizzare rispettivamente l'istruzione PEN OFF o STRIG OFF. L'esecuzione del programma diventa più rapida, dato che non è più necessario controllare questi due strumenti. Se si desidera sospendere la gestione degli eventi, permettendo però al programma di ricordare se la penna ottica o il joystick sono stati utilizzati durante la sospensione, è necessario utilizzare PEN STOP o STRIG STOP. Quando il programma esegue la successiva istruzione PEN ON o STRIG ON, passa alla subroutine corretta, se la penna ottica o il joystick sono stati attivati.

I VALORI DI POSIZIONE

Ovviamente, il programma deve essere in grado di determinare la posizione della penna ottica o del joystick al momento dell'intercettazione di un evento. Quando il programma conosce la posizione, è in grado di eseguire l'operazione corretta, ad esempio attivare un'opzione del menu oppure spostare un pezzo del gioco.

Per quanto riguarda la penna ottica, la funzione PEN può essere utilizzata con un numero che indichi le informazioni restituite. Si noti che PEN può restituire le coordinate di una posizione sia sotto forma di carattere (posizione della cella di carattere) che sotto forma grafica (locazione del pixel). Questa funzione può restituire la posizione in corrispondenza della quale è stata attivata la penna ottica o quella in cui è stata puntata l'ultima volta, che non sono necessariamente identiche; può inoltre leggere la posizione della penna ottica, persino quando quest'ultima non è stata attivata, quindi, un programma di disegno può consentire all'utente di attivare la penna ottica in corrispondenza del punto iniziale e quindi tracciare una linea di punti, mentre la penna si sposta. Questo tipo di operazione viene illustrata nel listato di seguito riportato.

```
SCREEN 1
PEN ON

DO
   IF PEN(0) = -1 THEN PSET (PEN(4), PEN(5))
LOOP UNTIL INKEY$ = CHR$(27)
```

Per quanto riguarda il joystick, la funzione STICK può essere utilizzata con un numero che indica la coordinata *x* o *y* restituita. Il joystick sfrutta un sistema di coordinate indipendente da quello utilizzato nella modalità video in uso. Joystick diversi variano nel risultato che restituiscono: alcuni restituiscono coordinate nell'intervallo da 1 a 80, altri possono utilizzare coordinate anche maggiori di 200. La funzione STICK restituisce soltanto i valori nell'intervallo da 1 a 200. Partendo dal presupposto che le variabili maxX% e maxY% contengano i valori massimi delle coordinate *x* e *y* nella modalità grafica in uso e che maxXJoy% e maxYJoy% contengano

invece i valori massimi restituiti, è possibile calcolare il fattore di scala per convertire le coordinate del joystick in quelle dello schermo, mediante le seguenti istruzioni:

```
xScala% = maxX% / maxXJoy%
yScala% = maxY% / maxYJoy%
```

Per convertire la coordinata del joystick in quella dello schermo, è sufficiente moltiplicare la prima per il fattore di scala. Ad esempio, per assegnare la posizione del joystick numero 1 alle variabili xPos% e yPos%, è necessario utilizzare le seguenti istruzioni:

```
xPos% = STICK(0)
yPos% = STICK(1)
```

Se poi si desidera utilizzare l'istruzione PUT per visualizzare un'immagine in corrispondenza della posizione indicata dal joystick, è sufficiente sfruttare la seguente istruzione:

```
PUT (xPos% * xScala%, yPos% * yScala%), grafArray%, PSET
```

A causa delle diversità nella progettazione e nella configurazione dei joystick, il programma deve permettere all'utenti di calibrarli, puntandoli in punti determinati dello schermo. Durante la calibrazione, il programma può utilizzare i valori restituiti da STICK per determinare il fattore di scala.

La funzione STRIG restituisce le informazioni relative ai pulsanti del joystick. Per indicare il pulsante da esaminare, è sufficiente inserire un numero nella chiamata a STRIG. Questa funzione risulta utile soprattutto se il programma contiene una subroutine a cui passa l'esecuzione quando viene premuto un pulsante. Utilizzando STRIG, la subroutine è in grado di determinare quale pulsante è stato premuto.

UTILIZZO DEL MOUSE COME PENNA OTTICA

Se al computer non è collegata una penna ottica, ma si dispone di un mouse con un driver installato, è possibile utilizzarlo più o meno come una penna. Sebbene non sia possibile puntare con un mouse direttamente allo schermo, la funzione PEN restituisce la riga e la colonna del carattere in corrispondenza del quale è posizionato il puntatore del mouse, se vengono premuti entrambi i pulsanti. L'utilizzo del mouse come penna ottica è utile per la verifica delle istruzioni, quali PRINT PEN(4), PEN(5), che si trovano in programmi che devono essere utilizzati con una penna ottica.

Bibliografia

Microsoft Press and Hardware Division of Microsoft Corporation. *Microsoft Mouse Programmer's Reference.* Redmond, Wash.: Microsoft Press, 1989.

Prata, Stephen, with Harry Henderson. *The Waite Group's Microsoft QuickBASIC Primer Plus.* Redmond, Wash.: Microsoft Press, 1990.

Schneider, David L. *Handbook of BASIC.* 3d ed. Boston, Mass.: Brady, 1988.

ON PEN GOSUB

Si consulti anche

PEN, PEN ON/OFF/STOP

Scopo

L'istruzione ON PEN GOSUB specifica la subroutine in corrispondenza della quale passa l'esecuzione, quando viene attivata la penna ottica; deve essere utilizzata quando si desidera che il programma risponda alla penna ottica, eseguendo una gestione degli eventi che effettui una determinata operazione.

Sintassi

```
ON PEN GOSUB {riga | etichetta}
```

riga o etichetta indicano il numero di riga o l'etichetta dove inizia la subroutine che deve essere eseguita.

Utilizzo

```
ON PEN GOSUB GestionePenna
```

Indica che viene eseguita la subroutine GestionePenna, quando viene attivata la penna ottica.

Descrizione

La gestione degli eventi relativa alla penna ottica può essere utilizzata per eseguire una determinata subroutine, quando viene riscontrata l'attivazione della penna ottica. Innanzitutto, è necessario richiamare l'istruzione ON PEN GOSUB per indicare la subroutine da eseguire, utilizzando quindi PEN ON per attivare l'intercettazione della penna ottica.

Errori

Se la penna ottica non è collegata oppure non è stata calibrata correttamente, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita".

Avvertenze

QBasic non risponde alla penna ottica, se nel proprio sistema è stato installato un driver per il mouse; quest'ultimo può essere utilizzato per emulare la penna ottica.

Esempio

Il programma di seguito riportato illustra la gestione degli eventi relativa alla penna ottica. Nell'istruzione ON PEN GOSUB viene specificato che deve essere eseguita la subroutine Attiva, quando la penna ottica viene attivata. La subroutine si limita a emettere un suono acustico, a stampare un messaggio e a uscire dal programma. Se si dispone di un mouse con due pulsanti, è sufficiente premerli entrambi contemporaneamente per richiamare la subroutine.

```
CLS
ON PEN GOSUB Attiva        'specifica la subroutine per la gestione della penna
                           'ottica
PEN ON                     'abilita la gestione degli eventi della penna ottica

DO
   LOCATE 1, 1
   PRINT "Attiva la penna ottica."
LOOP UNTIL INKEY$ = CHR$(27)          'esce dal ciclo quando l'utente preme
Esc
END

Attiva:                               'la penna ottica è stata attivata
    BEEP
    PRINT "La penna ottica è stata attivata."
    SYSTEM                            'termina il programma
    RETURN
```

ON STRIG GOSUB

Si consulti anche

STRIG, STRIG ON/OFF/STOP

Scopo

L'istruzione ON STRIG GOSUB specifica la subroutine in corrispondenza della quale passa l'esecuzione, quando viene premuto un particolare pulsante del joystick; deve essere utilizzata quando si desidera che il programma risponda alla pressione del pulsante del joystick, eseguendo una gestione degli eventi che effettui una determinata operazione.

Sintassi

```
ON STRIG GOSUB(pulsante) {riga | etichetta}
```

pulsante è il numero 0, 2, 4 o 6. 0 indica il pulsante inferiore del primo joystick, 2 specifica il pulsante inferiore del secondo, 4 indica il pulsante superiore del primo joystick, mentre 6 specifica il pulsante superiore del secondo.

riga o etichetta indicano il numero di riga o l'etichetta dove inizia la subroutine che deve essere eseguita.

Utilizzo

```
ON STRIG(4) GOSUB Fuoco
```

Indica che viene eseguita la subroutine Fuoco, quando viene premuto il pulsante superiore del primo joystick.

Descrizione

La gestione degli eventi relativa al joystick può essere utilizzata per eseguire una determinata subroutine, quando il programma individua che è stato premuto un determinato pulsante del joystick (QBasic supporta uno o due joystick, ognuno avente uno o due pulsanti). L'istruzione ON STRIG GOSUB indica a quale dei quattro pulsanti

si deve rispondere e la subroutine da eseguire. I numeri 0 e 4 fanno riferimento ai pulsanti inferiore e superiore del primo joystick; 2 e 6 invece fanno riferimento ai pulsanti inferiore e superiore del secondo joystick. Alcuni di questi dispositivi hanno uno o due pulsanti sulla base e uno sulla parte superiore dell'impugnatura, che viene abitualmente definito pulsante superiore. Per ulteriori informazioni sulla designazione dei pulsanti, si consulti la documentazione fornita insieme al joystick.

Dopo aver richiamato l'istruzione ON STRIG GOSUB, è necessario utilizzare STRIG ON per attivare l'intercettazione del pulsante del joystick.

Esempio

Il programma di seguito riportato consente di intercettare sia il pulsante inferiore che quello superiore del primo joystick. Quando viene premuto un pulsante, il programma esegue la subroutine indicata da ON STRIG GOSUB, la quale emette un segnale acustico e visualizza un messaggio. La subroutine legata al pulsante superiore, consente anche di uscire dal programma richiamando l'istruzione SYSTEM.

```
CLS
ON STRIG(0) GOSUB pulsanteInf        'specifica la subroutine per il pulsante
                                     'inferiore del joystick
ON STRIG(4) GOSUB pulsanteSup        'specifica la subroutine per il pulsante
                                     'superiore del joystick

STRIG(0) ON                    'abilita la gestione per il pulsante inferiore
STRIG(4) ON                    'abilita la gestione per il pulsante superiore

PRINT "Premi l'uno o l'altro dei pulsanti del joystick."
PRINT "La pressione del pulsante superiore causa la fine del programma."

LOCATE 4, 1
PRINT "Premi l'uno o l'altro dei pulsanti del joystick."
DO
LOOP UNTIL INKEY$ = CHR$(27)
END

pulsanteInf:                         'subroutine per il pulsante inferiore
  BEEP
  LOCATE 6, 1
  PRINT "Hai premuto il pulsante inferiore."
  RETURN

pulsanteSup:                         'subroutine per il pulsante superiore
  BEEP
  LOCATE 6, 1
  PRINT "Hai premuto il pulsante superiore."
  SYSTEM
```

PEN

Si consulti anche

ON PEN GOSUB, PEN ON/OFF/STOP

Scopo

La funzione PEN restituisce lo stato attuale della penna ottica; deve essere utilizzata per determinare la posizione della penna ottica sullo schermo.

Sintassi

```
PEN(numero)
```

numero è un intero nell'intervallo da 0 a 9.

Utilizzo

```
ultimoX% = PEN(1)
ultimoY% = PEN(2)
```

Assegna alle variabili ultimoX% ultimoY% le coordinate dei pixel della posizione in corrispondenza della quale è stata attivata la penna ottica.

Descrizione

La funzione PEN può essere utilizzata per determinare se la penna ottica è stata attivata, oppure se è in fase di attivazione. Questa funzione può essere utilizzata anche per individuare le coordinate dei grafici o del testo in corrispondenza dei quali la penna ottica è stata attivata o puntata. Prima di poter utilizzare l'istruzione PEN, è necessario attivare la gestione degli eventi mediante l'istruzione PEN ON.

Il numero da 0 a 9 tra parentesi, dopo la parola chiave PEN, indica le informazioni che la funzione restituisce. Nella Tabella 13.3 sono riportati i valori che PEN restituisce per ogni numero. La differenza tra le funzioni PEN(0) e PEN(3) è che la prima restituisce il valore vero (-1), se la penna è stata attivata dall'ultima chiamato alla funzione PEN, mentre la seconda restituisce il valore vero soltanto se la penna è attiva nel momento in cui viene richiamata PEN(3).

La funzione PEN può essere utilizzata in modalità grafica o testo; nel primo caso le funzioni PEN(1) e PEN(2) possono essere utilizzate per determinare in modo approssimativo le coordinate *x* e *y* del pixel in corrispondenza del quale è stata attivata l'ultima volta la penna ottica, mentre PEN(4) e PEN(5) per determinare le coordinate a cui è puntata. Il valore delle coordinate è approssimativo, dato che la maggior parte delle penne ottiche non dispone di una risoluzione paragonabile a quella dello schermo grafico. Ad esempio, la penna ottica potrebbe essere in grado di accedere soltanto alle coordinate *x* 0, 8, 16, 24, ... 320 e alle coordinate *y* 0, 2, 4, 6, ... 200. Una penna ottica puntata nel punto (238, 149) potrebbe restituire le coordinate 240 e 150.

In modalità testo, le funzioni PEN(6) e PEN(7) possono essere utilizzate per determinare la riga e la colonna di testo in cui la penna ottica è stata attivata l'ultima volta, mentre le funzioni PEN(8) e PEN(9) per determinare la riga e la colonna in cui la penna ottica è stata puntata l'ultima volta.

Avvertenze

QBasic non risponde alla penna ottica, se nel proprio sistema è stato installato un driver per il mouse; quest'ultimo può essere utilizzato per emulare la penna ottica.

Esempio

Il programma di seguito riportato, oltre a visualizzare tutte le informazioni fornite dalla funzione PEN, presenta anche un piccolo riquadro: premendovi la penna ottica sopra (oppure puntando al riquadro, se la penna è del tipo "touch-ring"), il programma termina.

Tabella 13.3 I valori restituiti dalla funzione PEN

Numero	Valore restituito
0	-1 se la penna ottica è stata attivata dalla più recente chiamata a PEN; in caso contrario 0.
1	la coordinata x del pixel in corrispondenza del quale la penna ottica è stata attivata l'ultima volta.
2	la coordinata y del pixel in corrispondenza del quale la penna ottica è stata attivata l'ultima volta.
3	-1 se la penna ottica è attiva; in caso contrario 0.
4	la coordinata x del pixel in corrispondenza del quale la penna ottica è stata puntata l'ultima volta.
5	la coordinata y del pixel in corrispondenza del quale la penna ottica è stata puntata l'ultima volta.
6	la riga di testo in corrispondenza della quale la penna ottica è stata attivata l'ultima volta.
7	la colonna di testo in corrispondenza della quale la penna ottica è stata attivata l'ultima volta.
8	la riga di testo in corrispondenza della quale la penna ottica è stata puntata l'ultima volta.
9	la colonna di testo in corrispondenza della quale la penna ottica è stata puntata l'ultima volta.

```
PEN ON
CLS
LOCATE 10, 35
PRINT "Tocca il riquadro per uscire dal programma."; CHR$(219)
'carattere riquadro
DO
   LOCATE 20, 1
   PRINT "La penna ottica ";
   IF PEN(0) THEN                        'verifica l'attivazione della penna
       PRINT "è  stata   attivata";
   ELSE PRINT "non  è stata attivata";
   END IF

   LOCATE 20, 40
   PRINT "Cambia lo stato attuale in ";
   IF PEN(3) THEN                        'verifica se la penna ottica è attiva
       PRINT "    giù";
   ELSE PRINT "non giù";
   END IF

   LOCATE 21, 1
   PRINT "Ultima attivazione a X: "; PEN(1);
   LOCATE 21, 40
   PRINT "Ultima attivazione a Y: "; PEN(2);

   LOCATE 22, 1
   PRINT "Ora a X: "; PEN(4);
   LOCATE 22, 40
   PRINT "Ora a Y: "; PEN(5);

   LOCATE 23, 1
   PRINT "Attivata alla riga: "; PEN(6);
   LOCATE 23, 40
```

```
    PRINT "Attivata alla colonna: "; PEN(7);

    LOCATE 24, 1
    PRINT "Ora a riga: "; PEN(8);
    LOCATE 24, 40
    PRINT "Ora a colonna: "; PEN(9);

    'verifica che la penna ottica sia attivata sul riquadro
    IF (PEN(6) = 10) AND (PEN(7) = 70) THEN SYSTEM
LOOP
```

PEN ON/OFF/STOP

Si consulti anche

ON PEN GOSUB, PEN

Scopo

Le istruzioni PEN consentono di attivare, disattivare o sospendere la gestione degli eventi relativi alla penna ottica. PEN ON consente inoltre di attivare la funzione PEN, che può essere sfruttata per ottenere alcune informazioni riguardanti lo stato della penna ottica.

Sintassi

```
PEN ON
PEN OFF
PEN STOP
```

Utilizzo

```
PEN ON
```

Attiva sia la gestione degli eventi relativi alla penna ottica che la funzione PEN.

Descrizione

L'istruzione PEN ON può essere utilizzata per tenere traccia dei movimenti e dello stato (attivato o non attivato) della penna ottica. Se quest'ultima è stata attivata ed è stata richiamata un'istruzione ON PEN GOSUB, il programma passa alla subroutine in essa specificata.

L'istruzione PEN OFF disabilita la gestione degli eventi: il programma non passa alla subroutine, quando la penna ottica viene attivata, e la funzione PEN non restituisce alcuna informazione relativa alla penna. Quando viene ripristinata la gestione degli eventi, il programma non ricorda gli eventi che si sono verificati quando questa caratteristica era disattivata.

L'istruzione PEN STOP sospende la gestione degli eventi: il programma continua a prendere nota dell'attività della penna ottica, ma non esegue la subroutine specificata nell'istruzione ON PEN GOSUB. Quando viene nuovamente eseguita un'istruzione PEN ON, la subroutine viene eseguita soltanto se la penna ottica era attivata al momento della sospensione della gestione degli eventi.

Il programma esegue un'istruzione PEN STOP implicita ogniqualvolta esegue la subroutine specificata con ON PEN GOSUB; questo garantisce che un'altra ricorrenza dell'evento non provochi un'esecuzione ricorsiva della subroutine. Quando il programma

individua un'istruzione RETURN e ritorna al programma principale, imposta nuovamente la gestione degli eventi, eseguendo un'istruzione PEN ON implicita.

Suggerimenti

Dato che la gestione degli eventi relativi alla penna ottica rallenta leggermente l'esecuzione del programma, è bene disattivarla quando non è previsto un input da parte dell'utente con tale dispositivo.

QBasic non risponde alla penna ottica se nel proprio sistema è stato installato un driver per il mouse; quest'ultimo può essere utilizzato per emulare la penna ottica.

Esempio

L'esempio di seguito riportato utilizza l'istruzione PEN ON per attivare la gestione degli eventi relativi alla penna ottica. Il programma esegue la subroutine Attiva, quando la penna ottica viene attivata la prima volta. La subroutine assegna 1 alla variabile gestione%, che fa in modo che il programma esegua l'istruzione PEN STOP individuata nel ciclo principale; la gestione degli eventi viene attivata nuovamente quando si esce dal ciclo premendo un tasto. Il programma ricorda se la penna è stata attivata dopo PEN STOP e quindi, nel caso, esegue di nuovo Attiva. Si noti che una chiamata a PEN STOP non è efficace, se si trova all'interno di una subroutine, dato che il programma richiama sempre PEN ON, quando esce dalla subroutine stessa.

```
CLS
ON PEN GOSUB Attiva
PEN ON                     'attiva la gestione degli eventi della penna ottica
gestione% = 0              'flag segnalante che si è verificato un evento

DO
   LOCATE 1, 1
   PRINT "Attiva la penna ottica."
   IF gestione% = 1 THEN PEN STOP
LOOP UNTIL INKEY$ <> ""         'esce se l'utente preme un tasto qualsiasi
PEN ON                          'esegue la subroutine Attiva se la penna
ottica è stata attivata
END

Attiva:                         'la penna ottica è stata attivata
    BEEP
    PRINT "La subroutine Attiva sarà eseguita più di una volta"
    PRINT "se attivi la penna ottica e quindi premi un tasto."
    gestione% = 1
    RETURN
```

STICK

Si consulti anche

ON STRIG GOSUB, STRIG, STRIG ON/OFF/STOP

Scopo

La funzione STICK restituisce le coordinate *x* e *y* relative al punto sul quale è stato puntato il joystick.

Sintassi

```
STICK(numero)
```

numero è un intero nell'intervallo da 0 a 3.

Utilizzo

```
stick1X% = STICK(0)
```

Assegna la coordinata *x* del primo joystick alla variabile stick1X%.

Descrizione

La funzione STICK restituisce le informazioni relative alle coordinate *x* e *y* del joystick. I valori restituiti dipendono dal numero specificato; nella Tabella 13.4 sono riportati i numeri e i valori corrispondenti restituiti dalla funzione.

Joystick diversi variano nel risultato che restituiscono: alcuni restituiscono coordinate nell'intervallo da 1 a 80, altri possono utilizzare coordinate anche maggiori di 200. La funzione STICK restituisce soltanto i valori nell'intervallo da 1 a 200, indipendentemente dalla modalità video attiva. Se si desidera disegnare (o ridisegnare) un cursore che deve essere spostato con il joystick, è necessario rapportare le coordinate del joystick con quelle dello schermo. Ad esempio, se si utilizza uno schermo grafico di 640 per 480 pixel e un joystick, le cui coordinare vanno da 1 a 200, il valore x restituito dalla funzione STICK deve essere moltiplicato per 640/200, oppure per 3.2, mentre il valore y deve essere moltiplicato per 480/200 o per 2.4.

La funzione STRIG può essere utilizzata per determinare se è stato premuto un pulsante del joystick.

Commenti

L'intervallo da 1 a 200 limita la risoluzione delle coordinate del joystick e quindi anche la sua precisione. Il risultato ottenuto permette in ogni caso all'utente di effettuare la selezione dei menu e di utilizzare giochi di simulazione.

Se al sistema non è collegato nessun joystick, qualsiasi chiamata alla funzione STICK restituisce il valore 0 e non produce alcun messaggio di errore.

Esempio

Il programma di seguito riportato disegna il mirino di un cannone che può essere spostato sullo schermo mediante il joystick. Premendo il pulsante inferiore, il programma utilizza la funzione STICK per visualizzare nell'angolo inferiore destro le coordinate del joystick in uso. Premendo il pulsante superiore, il programma termina. I valori delle variabili di scala scalaX e ScalaY possono essere modificati, se il joystick è in grado di restituire valori che si trovano al di fuori dell'intervallo da 1 a 80.

```
CLS
PRINT "Muovi il mirino del cannone con il joystick numero 1."
PRINT "Premi il bottone inferiore per vedere le coordinate."
PRINT "Premi il bottone superiore per uscire dal programma."
PRINT "<Premi un tasto qualsiasi per incominciare.>"
DO
LOOP UNTIL INKEY$ <> ""
```

Tabella 13.4 I valori restituiti dalla funzione STICK

Numero	Valore restituito
0	coordinata x del primo joystick
1	coordinata y del primo joystick
2	coordinata x del secondo joystick
3	coordinata y del secondo joystick

```
ON STRIG(0) GOSUB MostraCoord                    'mostra le coordinate attuali
ON STRIG(4) GOSUB UscitaProg                     'esce dal programma
STRIG(0) ON          'gestisce il pulsante inferiore del primo joystick
STRIG(4) ON          'gestisce il pulsante superiore del primo joystick

'variabili globali
vecchioX = 0         'coordinata x dell'ultimo mirino disegnato
vecchioY = 0         'coordinata y dell'ultimo mirino disegnato
nuovoX = 0           'coordinata x del nuovo mirino
nuovoY = 0           'coordinata y del nuovo mirino

'il fattore di scala assume che il joystick abbia un raggio di 80 in ogni
'direzione
scalaX = 4           'fattore di scala per x in CGA
scalaY = 2.5         'fattore di scala per y in CGA

SCREEN 1             'modalità schermo in CGA

DO                   'cicla fino a quando il pulsante superiore viene premuto
   GOSUB Mirino      'chiama il sottoprogramma Mirino
LOOP

MostraCoord:   'mostra le coordinate quando il pulsante inferiore è premuto
      LOCATE 24, 20
      PRINT "X ="; STICK(0);
      LOCATE 20, 30
      PRINT "Y ="; STICK(1);
      RETURN

Mirino:                                          'disegna il mirino
      nuovoX = STICK(0) * scalaX
      nuovoY = STICK(1) * scalaY

      'cancella il vecchio mirino
      LINE (vecchioX, vecchioY)-(vecchioX + 5, vecchioY), 0
      LINE (vecchioX, vecchioY)-(vecchioX - 5, vecchioY), 0
      LINE (vecchioX, vecchioY)-(vecchioX, vecchioY + 3), 0
      LINE (vecchioX, vecchioY)-(vecchioX, vecchioY - 3), 0

      'disegna il nuovo mirino
      LINE (nuovoX, nuovoY)-(nuovoX + 5, nuovoY), 1
      LINE (nuovoX, nuovoY)-(nuovoX - 5, nuovoY), 1
      LINE (nuovoX, nuovoY)-(nuovoX, nuovoY + 3), 1
      LINE (nuovoX, nuovoY)-(nuovoX, nuovoY - 3), 1

      vecchioX = nuovoX
      vecchioY = nuovoY
      RETURN

UscitaProg:
      SYSTEM
      RETURN
```

STRIG

Si consulti anche

ON STRIG GOSUB, STRIG ON/OFF/STOP

Scopo

La funzione STRIG restituisce lo stato di un pulsante del joystick. Queste informazioni possono essere utilizzate per elaborare la selezione di un menu da parte di un utente, oppure per eseguire un comando di "fuoco" in un gioco.

Sintassi

```
STRIG(numero)
```

numero è un intero nell'intervallo da 0 a 7.

Utilizzo

```
IF STRIG(0) GOSUB FuocoA
```

Se l'utente premere il pulsante inferiore del joystick 1, il programma esegue la subroutine FuocoA.

Descrizione

La funzione STRIG può essere utilizzata con un numero da 0 a 7 per determinare se è stato premuto o è attualmente premuto un determinato pulsante del joystick dall'ultima chiamata a STRIG con lo stesso numero. Nella Tabella 13.5 sono riportati i numeri e i relativi valori restituiti da STRIG. In ogni caso, QBasic restituisce il valore -1 (vero), se il pulsante viene premuto e 0 se il pulsante non viene premuto. QBasic supporta due joystick, ognuno con due pulsanti.

Come alternativa all'utilizzo di IF o di un'altra istruzione condizionale per verificare il valore di una chiamata STRIG, è possibile utilizzare l'istruzione ON STRIG GOSUB per impostare la gestione degli eventi relativa al joystick, che consente di eseguire una determinata subroutine quando l'utente preme un particolare pulsante.

Se il joystick a cui si fa riferimento in una chiamata alla funzione STRIG non è installato, la chiamata restituisce il valore 0 (falso), ma QBasic non visualizza alcun messaggio di errore.

Esempio

Il programma di seguito riportato indica quale pulsante è stato premuto sul primo joystick e fa emettere un suono all'altoparlante del computer.

```
DO
   IF STRIG(0) THEN PRINT "Pulsante inferiore premuto!"; CHR$(7)
   IF STRIG(4) THEN PRINT "Pulsante superiore premuto!"; CHR$(7)
LOOP UNTIL INKEY$ <> ""       'esce dal ciclo quando l'utente preme un tasto
```

Tabella 13.5 I valori restituiti dalla funzione STRING

Numero	Valore restituito
0	-1 se è stato premuto il pulsante inferiore del joystick 1 dall'ultima chiamata a STRIG(0); in caso contrario 0.
1	-1 se il pulsante inferiore del joystick 1 viene premuto in quel momento; in caso contrario 0.
2	-1 se è stato premuto il pulsante inferiore del joystick 2 dall'ultima chiamata a STRIG(2); in caso contrario 0.
3	-1 se il pulsante inferiore del joystick 2 viene premuto in quel momento; in caso contrario 0.
4	-1 se è stato premuto il pulsante superiore del joystick 1 dall'ultima chiamata a STRIG(4); in caso contrario 0.
5	-1 se il pulsante superiore del joystick 1 viene premuto in quel momento; in caso contrario 0.
6	-1 se è stato premuto il pulsante superiore del joystick 2 dall'ultima chiamata a STRIG(6); in caso contrario 0.
7	-1 se il pulsante superiore del joystick 2 viene premuto in quel momento; in caso contrario 0.

STRIG ON/OFF/STOP

Si consulti anche

ON STRIG GOSUB, STRIG

Scopo

Le istruzioni STRIG consentono di attivare, disattivare o sospendere la gestione degli eventi relativi al joystick.

Sintassi

```
STRIG(pulsante) ON
STRIG(pulsante) OFF
STRIG(pulsante) STOP
```

pulsante è il numero 0, 2, 4 o 6. 0 indica il pulsante inferiore del primo joystick, 2 specifica il pulsante inferiore del secondo, 4 indica il pulsante superiore del primo joystick, mentre 6 specifica il pulsante superiore del secondo.

Utilizzo

```
STRIG(0) ON
```

Attiva la gestione del pulsante inferiore sul primo joystick.

Descrizione

L'istruzione STRIG ON può essere utilizzata per tenere traccia dello stato (premuto o non premuto) del pulsante del joystick indicato dal numero fornito. Nella Tabella

13.6 sono riportati i numeri e i pulsanti ai quali corrispondono. Se è stato premuto un pulsante, il programma esegue la subroutine specificata nell'istruzione ON STRIG GOSUB.

L'istruzione STRIG OFF disabilita la gestione degli eventi del pulsante del joystick specificato; per ripristinarla, è necessario eseguire un'istruzione STRIG ON. Il programma non ricorda i tasti premuti quando la gestione degli eventi era disattivata.

L'istruzione STRIG STOP sospende la gestione degli eventi: il programma continua a prendere nota dell'attività del joystick, ma non esegue la subroutine specificata nell'istruzione ON STRIG GOSUB. Quando viene nuovamente eseguita un'istruzione STRIG ON, la subroutine viene eseguita soltanto se il pulsante era stato premuto, al momento della sospensione della gestione degli eventi.

Il programma esegue un'istruzione STRIG STOP implicita ogniqualvolta viene eseguita la subroutine indicata da ON STRIG GOSUB. Questo garantisce che un altra ricorrenza dell'evento non provochi un'esecuzione ricorsiva della subroutine. Quando il programma individua un'istruzione RETURN e ritorna al programma principale, imposta nuovamente la gestione degli eventi eseguendo un'istruzione STRIG ON implicita.

Tabella 13.6 I numeri e i pulsanti corrispondenti per i joystick

Numero	Pulsante
0	Pulsante inferiore del primo joystick
2	Pulsante inferiore del secondo joystick
4	Pulsante superiore del primo joystick
6	Pulsante superiore del secondo joystick

Suggerimenti

Dato che la gestione degli eventi relativi al joystick rallenta leggermente l'esecuzione del programma, è bene disattivarla quando non è previsto un input da parte dell'utente con questo dispositivo.

Esempio

Il programma di seguito riportato illustra la gestione degli eventi del joystick, che è stata impostata in modo tale che, se viene premuto il pulsante inferiore o quello superiore, vengono eseguite le subroutine Inferiore o Superiore. Se viene premuto il pulsante inferiore, Inferiore utilizza l'istruzione STRIG OFF per disattivare l'intercettazione di quel pulsante, in modo che la subroutine non venga eseguita se esso viene nuovamente premuto. Se invece viene premuto il pulsante superiore, Superiore utilizza un'istruzione STRIG(0) ON per attivare nuovamente l'intercettazione del pulsante inferiore, in modo che il programma possa nuovamente rispondere a quel pulsante (quindi la disattiva nuovamente).

```
CLS

ON STRIG(0) GOSUB Inferiore     'imposta la gestione per il pulsante inferiore
                                'del joystick 1
ON STRIG(4) GOSUB Superiore     'imposta la gestione per il pulsante superiore
                                'del joystick 1

STRIG(0) ON                     'abilita la gestione del pulsante inferiore
STRIG(4) ON                     'abilita la gestione del pulsante superiore

LOCATE 1, 1
PRINT "Premi uno qualsiasi dei pulsanti del joystick o premi un tasto
qualsiasi per uscire."
PRINT STRING$(56, "_")

VIEW PRINT 3 TO 7         'imposta il viewport di testo da riga 3 a riga 7
DO
LOOP UNTIL INKEY$ <> ""   'cicla fino a quando l'utente preme un tasto
END

Inferiore:                               'subroutine per il pulsante inferiore
       BEEP
       PRINT "Hai premuto il pulsante inferiore."
       PRINT "Il pulsante inferiore non sarà intercettato di nuovo"
       PRINT "fino a quando non sarà premuto il pulsante superiore."
       PRINT STRING$(55, "_")
       STRIG(0) OFF       'disabilita la gestione del pulsante inferiore
       RETURN

Superiore:                'subroutine per il pulsante superiore
       BEEP
       STRIG(0) ON        'riabilita la gestione del pulsante inferiore
       PRINT "Hai premuto il pulsante superiore."
       PRINT STRING$(55, "_")
       RETURN
```

Parte terza

Dispositivi

Capitolo 14

La tastiera

INTRODUZIONE

Le istruzioni di QBasic che gestiscono l'input da tastiera forniscono un collegamento essenziale tra l'utente e il programma. La maggiore parte dei PC IBM e compatibili dispone di tastiere che comprendono 10 o 12 tasti funzione. Le istruzioni di gestione degli eventi della tastiera consentono di impostare una routine che viene richiamata ogniqualvolta l'utente preme un determinato tasto; molti programmi, ad esempio, visualizzano una schermata di aiuto ogni volta che l'utente preme il tasto F1.

In questo capitolo vengono esaminate le istruzioni di gestione della tastiera e vengono offerti alcuni suggerimenti relativi al loro utilizzo. Nella Tabella 14.1 è riportato un elenco di queste istruzioni.

TASTI RAPIDI

Coloro che hanno già utilizzato GW-BASIC o BASIC sono in grado probabilmente di riconoscere l'elenco che indica i tasti funzione e le corrispondenti istruzioni abbreviate visualizzato nella riga inferiore dello schermo; questi tasti vengono definiti *tasti rapidi* e consentono di utilizzare i tasti funzione per immettere le istruzioni più comuni, premendo semplicemente un tasto. L'istruzione KEY di QBasic permette di impostare questi tasti per i propri programmi.

Ad esempio, utilizzando l'istruzione di seguito riportata è possibile trasformare F1 in un tasto rapido che visualizzza la stringa "AIUTO".

```
KEY 1, "AIUTO"
```

Ora, ogniqualvolta viene premuto F1 durante l'esecuzione del programma, qualsiasi funzione o istruzione di input dalla tastiera (comprese INKEY$, INPUT e LINE INPUT) accetta la parola "AIUTO" come input.

Tabella 14.1 Le istruzioni di QBasic relative alla tastiera

Istruzione	Descrizione
KEY	Imposta i tasti rapidi.
KEY(n) ON/OFF/STOP	Attiva, disattiva o sospende la gestione degli eventi della tastiera impostata dall'istruzione ON KEY(n) GOSUB.
ON KEY(n) GOSUB	Imposta la gestione degli eventi della tastiera.

L'istruzione KEY LIST deve essere utilizzata per visualizzare l'impostazione in uso dei tasti rapidi, che appaiono uno alla volta sullo schermo. L'istruzione KEY ON visualizza le impostazioni dei tasti rapidi sulla riga inferiore dello schermo, mentre KEY OFF cancella tale visualizzazione. I tasti rapidi possono essere utilizzati anche se non sono visualizzati.

Anche se sembrano simili alle istruzioni di gestione degli eventi KEY(n) ON e KEY(n) OFF, le istruzioni KEY ON e KEY OFF vengono applicate soltanto alla visualizzazione dei tasti rapidi.

GESTIONE DEGLI EVENTI DELLA TASTIERA

QBasic fornisce una serie di istruzioni che consentono di attivare e di gestire l'intercettazione degli eventi della tastiera. È possibile intercettare un determinato tasto, in modo che il programma esegua una subroutine ogniqualvolta l'utente lo preme. Ad esempio, questa caratteristica potrebbe essere utilizzata per visualizzare un aiuto relativo al contesto ogni volta che l'utente preme il tasto F1. QBasic consente di intercettare i tasti funzione, i tasti di direzione e le combinazioni di tasti definiti dall'utente.

La gestione degli eventi può essere impostata richiamando innanzitutto l'istruzione ON KEY(n) GOSUB, che indica il nome della subroutine in corrispondenza della quale deve passare l'esecuzione, quando l'utente preme il tasto specificato. In seguito è necessario attivare la gestione degli eventi, richiamando l'istruzione KEY(n) ON.

Le istruzioni di seguito riportate impostano una gestione degli eventi e la attivano per eseguire la subroutine che inizia all'etichetta MostraAiuto, ogniqualvolta l'utente preme il tasto F1.

```
ON KEY(1) GOSUB MostraAiuto
KEY(1) ON
```

Bibliografia

Norton, Peter, and Richard Wilton. *The New Peter Norton Programmer's Guide to the IBM PC & PS/2.* Redmond, Wash.: Microsoft Press, 1988.

Prata, Stephen, with Harry Henderson. *The Waite Group's Microsoft QuickBASIC Primer Plus.* Redmond, Wash.: Microsoft Press, 1990

KEY

Si consulti anche

KEY(n) ON/OFF/STOP, ON KEY(n) GOSUB

Scopo

L'istruzione **KEY** consente di definire un tasto rapido (ovvero di assegnare una stringa a un tasto funzione) e di visualizzarlo; può essere utilizzata per fornire all'utente un metodo per generare una lunga sequenza di pressioni di tasti, premendone uno solo.

Sintassi

```
KEY numero, stringa
KEY LIST
KEY ON
KEY OFF
```

numero è il numero di un tasto funzione; i numeri corretti sono da 1 a 10, 30 (per F11) e 31 (per F12).

stringa è un'espressione stringa lunga da 1 a 15 caratteri; una stringa nulla disabilita il tasto rapido corrispondente.

Utilizzo

```
KEY 1, "USCIRE" + CHR$(13)
```

Imposta il tasto funzione F1 come tasto rapido che genera la parola USCIRE e un ritorno a capo.

```
KEY LIST
```

Visualizza le assegnazioni di stringa relative a tutti i tasti rapidi.

Descrizione

L'istruzione **KEY** consente di gestire i tasti rapidi, che sono tasti funzione da premere per produrre stringhe. Per creare o modificare i tasti rapidi, è necessario inserire dopo la parola chiave **KEY** un numero di tasto funzione e una stringa. Il primo parametro indica il tasto rapido: devono essere utilizzati i numeri da 1 a 10 per i tasti funzione da F1 a F10 e 30 e 31 per i tasti F11 e F12 sulle tastiere con 101 tasti; il parametro stringa deve essere un'espressione stringa con un massimo di 15 caratteri. Dopo aver utilizzato l'istruzione **KEY** per effettuare un'assegnazione, è sufficiente premere quel tasto funzione per creare la stringa corrispondente, proprio come se l'utente lo avesse immesso da tastiera.

Tutte le impostazioni attive dei tasti rapidi possono essere visualizzate mediante l'istruzione **KEY LIST**, che fa apparire un elenco lungo 12 righe (una per ogni tasto funzione) con tutti i 15 caratteri di ogni impostazione.

L'istruzione **KEY ON** visualizza i primi sei caratteri delle primi 10 impostazioni dei tasti rapidi sulla riga inferiore dello schermo, mentre non vengono visualizzate quelle relative ai tasti funzione F11 e F12. L'istruzione **KEY OFF** disabilita la visualizzazione dei tasti nella riga inferiore dello schermo, ma non impedisce di utilizzarli.

Commenti

Il parametro stringa può comprendere i caratteri di controllo quali CHR$(13) e il tasto INVIO. Per definire una combinazione di tasti adotta all'utilizzo con una gestione degli eventi della tastiera, è possibile utilizzare una variante dell'istruzione KEY.

Errori

In un'istruzione KEY che indica un numero e una stringa, se il parametro numerico non è compreso nell'intervallo da 1 a 10 e non è uguale a 30 o 31, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita".

Esempio

Il programma di seguito riportato utilizza l'istruzione KEY per fornire agli utenti alcuni tasti rapidi. Viene visualizzato un menu contenente alcune opzioni che possono essere selezionate premendo un tasto funzione o digitando una parola.

```
'imposta i valori stringa dei tasti
KEY 1, "Aggiunge" + CHR$(13)          'F1 - Aggiunge i record
KEY 2, "Modifica" + CHR$(13)          'F2 - Modifica il record
KEY 3, "Uscita" + CHR$(13)            'F3 - Esce dal programma

MenuPrincipale:
    CLS
    PRINT : PRINT : PRINT
    PRINT "<Aggiunge>......Aggiunge i record......[F1]"
    PRINT "<Modifica>......Modifica i record......[F2]"
    PRINT "<Uscita>........Esce dal programma.....[F3]"
    PRINT
    PRINT "Immettere una parola e premere Invio o premere un tasto funzione: ";
    LINE INPUT selezione$

    SELECT CASE UCASE$(selezione$)
        CASE "Aggiunge"
              PRINT : PRINT : PRINT
              PRINT "Aggiunge i record."

        CASE "Modifica"
              PRINT : PRINT : PRINT
              PRINT "Modifica i record."

        CASE "Uscita"
              PRINT : PRINT : PRINT
              PRINT "Esce dal programma."

        CASE ELSE
              PRINT : PRINT : PRINT
              PRINT "Selezione non valida - Riprova."
              BEEP
              SLEEP 1
              GOTO MenuPrincipale
    END SELECT
```

KEY(n) ON/OFF/STOP

Si consulti anche

KEY, ON KEY(n) GOSUB

Scopo

L'istruzione KEY(n) attiva, disattiva o sospende una gestione degli eventi per determinati tasti. Questa caratteristica deve essere utilizzata per indicare al programma di eseguire una subroutine, quando l'utente preme il tasto specificato.

Sintassi

```
KEY(n) ON
KEY(n) OFF
KEY(n) STOP
```

n è un numero che corrisponde a un tasto funzione, a un tasto di direzione oppure a un tasto definito dall'utente; i numeri corretti sono da 0 a 25, 30 e 31. Se n è 0, viene attivata la gestione di tutti i tasti.

Utilizzo

```
ON KEY(1) GOSUB OttenereAiuto
KEY(1) ON
```

Attiva la gestione degli eventi per il tasto funzione F1, in modo che l'esecuzione passi a OttenereAiuto, quando l'utente preme questo tasto.

Descrizione

L'istruzione KEY(n) ON deve essere utilizzata per attivare la gestione degli eventi per un particolare tasto; in questo caso il programma controlla, dopo ogni istruzione, che il tasto specificato sia stato premuto e in caso positivo esegue la routine indicata nell'istruzione KEY(n) ON GOSUB.

Nella Tabella 14.2 è riportato un elenco dei numeri che possono essere utilizzati nelle istruzioni KEY(n).

I tasti definiti dall'utente (i numeri da 15 a 25) possono essere definiti utilizzando la seguente variante dell'istruzione KEY:

```
KEY numero, CHR$(flagtastiera) + CHR$(scan)
```

numero indica un tasto definito dall'utente per la gestione degli eventi della tastiera e deve essere compreso nell'intervallo da 15 a 25.

flagtastiera è una combinazione di uno qualsiasi dei valori esadecimali riportati nella Tabella 14.3. Questi valori indicano lo stato dei diversi tasti "speciali" presenti sulla tastiera. La somma di due o più di questi valori può essere sfruttata per indicare i diversi tasti speciali che devono essere premuti contemporaneamente, prima che possa essere intercettato un tasto definito dall'utente. Ad esempio, il parametro &H04 + &H08 indica che è necessario premere sia CTRL che ALT. QBasic non effettua alcuna distinzione tra gli stessi tasti speciali a sinistra o a destra, quindi i codici &H01, &H02 e &H03 vengono gestiti in modo identico.

scan è il codice della tastiera di uno degli 83 tasti standard. Nella Tabella 14.4 è riportato l'elenco completo dei codici della tastiera per i PC IBM.

L'istruzione KEY(n) ON attiva la gestione degli eventi per il tasto indicato.

L'istruzione KEY(n) OFF disattiva invece tale caratteristica. Non si verifica alcuna gestione degli eventi, fino a quando il programma non esegue un'altra istruzione

Tabella 14.2 I numeri utilizzabili con KEY(n)

Numeri dei tasti	Tasti
0	Tutti i tasti riportati in questa tabella
da 1 a 11	I tasti funzione da F1 a F10
11	Freccia verso l'alto
12	Freccia sinistra
13	Freccia destra
14	Freccia verso il basso
da 15 a 25	Tasti definiti dall'utente
da 30 a 31	Tasti funzione F11 e F12

Tabella 14.3 I valori esadecimali per i tasti speciali

Valore	Tasti speciali
&H00	Nessuno
&H01 a &H03	Uno o entrambi i tasti Maiusc
&H04	Ctrl
&H08	Alt
&H20	Bloc Num
&H40	Bloc Maiusc
&H80	I tasti estesi su una tastiera di 101 tasti

KEY(n) ON. Il programma non è in grado di ricordare alcun evento che si è verificato quando la funzione di intercettazione era disattivata.

L'istruzione KEY(n) STOP sospende la gestione degli eventi relativa al tasto specificato. Non si verifica alcuna gestione degli eventi, fino a quando il programma non esegue un'altra istruzione KEY(n) ON; tuttavia, se si verifica un evento mentre la funzione di intercettazione è sospesa, il programma lo ricorda e, quando viene eseguita nuovamente l'istruzione KEY(n) ON, passa alla subroutine corretta.

Il programma esegue un'istruzione KEY(n) STOP implicita, ogniqualvolta viene eseguita la subroutine indicata da ON KEY(n) GOSUB. Questo garantisce che un'altra ricorrenza dell'evento non provochi l'esecuzione ricorsiva della subroutine. Quando il programma incontra un'istruzione RETURN e ritorna al programma principale, imposta nuovamente la gestione degli eventi della tastiera, eseguendo un'istruzione KEY(n) ON implicita.

Errori

Se il valore del numero del tasto specificato non si trova nell'intervallo da 0 a 25 e non è 30 o 31, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita".

Tabella 14.4 I codici della tastiera per i PC IBM

Tasto	Codice	Tasto	Codice
Esc	&H01	I o \	&H2B
! o 1	&H02	Z	&H2C
@ o 2	&H03	X	&H2D
# o 3	&H04	C	&H2E
$ o 4	&H05	V	&H2F
% o 5	&H06	B	&H30
^ o 6	&H07	N	&H31
& o 7	&H08	M	&H32
* o 8	&H09	< o ,	&H33
(o 9	&H0A	> o .	&H34
) o 0	&H0B	? o /	&H35
- o _	&H0C	MAIUSC DESTRA	&H36
+ o =	&H0D	STAMP o *	&H37
FRECCIA SINISTRA	&H0E	ALT	&H38
tab	&H0F	BARRA SPAZIATRICE	&H39
Q	&H10	BLOC MAIUSC	&H3A
W	&H11	F1	&H3B
E	&H12	F2	&H3C
R	&H13	F3	&H3D
T	&H14	F4	&H3E
Y	&H15	F5	&H3F
U	&H16	F6	&H40
I	&H17	F7	&H41
O	&H18	F8	&H42
P	&H19	F9	&H43
{ o [	&H1A	F10	&H44
} o]	&H1B	BLOC NUM	&H45
INVIO	&H1C	BLOC SCORR	&H46
CTRL	&H1D	HOME o 7	&H47
A	&H1E	FRECCIA IN ALTO o 8	&H48
S	&H1F	PAGSU o 9	&H49
D	&H20	- (tastierino numerico)	&H4A
F	&H21	FRECCIA SINISTRA o 4	&H4B
G	&H22	5	&H4C
H	&H23	FRECCIA DESTRA o 6	&H4D
J	&H24	+ (tastierino numerico)	&H4E
K	&H25	FINE o 1	&H4F
L	&H26	FRECCIA IN BASSO o 2	&H50
: o ;	&H27	PAGGIU o 3	&H51
“ o ‘	&H28	INS o 0	&H52
~ o ‘	&H29	CANC o .	&H53
MIUSC+SINISTRA	&H2A		

Suggerimenti

La gestione degli eventi rallenta l'esecuzione dei programmi, quindi è meglio non utilizzarla, a meno che non sia proprio indispensabile.

Esempio

Si consulti l'esempio riportato alla voce ON KEY(N) GOSUB.

ON KEY(n) GOSUB

Si consulti anche

KEY, KEY(n) ON/OFF/STOP

Scopo

L'istruzione ON KEY(n) GOSUB indica la subroutine che viene eseguita ogniqualvolta l'utente preme un determinato tasto; può essere utilizzata per impostare la gestione degli eventi della tastiera, ad esempio in modo che il programma stampi un messaggio di aiuto, quando l'utente preme il tasto F1.

Sintassi

```
ON KEY(n) GOSUB {riga | etichetta}
```

n è un numero che corrisponde a un tasto funzione, a un tasto di direzione oppure a un tasto definito dall'utente; i numeri corretti sono da 0 a 25, 30 e 31.

riga o etichetta indicano la prima riga di programma della subroutine di gestione degli eventi.

Utilizzo

```
ON KEY(1) GOSUB MostraAiuto
```

Indica che viene eseguita la subroutine MostraAiuto, quando l'utente preme il tasto funzione F1.

Descrizione

L'istruzione ON KEY(n) GOSUB deve essere utilizzata per impostare una gestione degli eventi che esegua una subroutine, quando viene premuto un determinato tasto. In questa istruzione è possibile specificare soltanto un tasto funzione, un tasto di direzione o un tasto definito dall'utente. Quando viene premuto il tasto indicato, il programma esegue la subroutine.

Quando quest'ultima inizia, il programma esegue un'istruzione KEY(n) STOP implicita, in modo che un'altra gestione degli eventi non provochi un'esecuzione ricorsiva della subroutine. Tuttavia se si è verificato un altro evento durante l'esecuzione della subroutine, il programma lo ricorda e quando la subroutine termina esegue un'istruzione KEY(n) ON (a meno che la subroutine non abbia eseguito esplicitamente un'istruzione KEY(n) OFF), eseguendo immediatamente l'evento che era sospeso.

La routine specificata può terminare con un'istruzione RETURN che trasferisce il controllo a una riga specificata del programma. Tuttavia, questa versione dell'istruzione RETURN può provocare gravi problemi, dato che la gestione degli eventi potrebbe aver interrotto un'operazione fondamentale; inoltre essa non può essere utilizzata con un numero specificato di riga o con un'etichetta per passare a un sottoprogramma, a una funzione o a una funzione DEF FN.

QBasic controlla la gestione degli eventi della tastiera nel seguente ordine.

1. Il tasto CTRL+STAMP che attiva o disattiva l'eco sulla stampante in linea; ridefinire questo tasto come definito dall'utente non ha alcun effetto.
2. I tasti funzione e quelli di direzione; la ridefinizione di questi tasti non influenza la gestione degli eventi di QBasic, dato che sono predefiniti.
3 I tasti definiti dall'utente.

L'istruzione ON KEY(n) GOSUB è in grado di intercettare qualsiasi combinazione di tasti, compresi CTRL+INTERR e CTRL+ALT+CANC. Quindi, è possibile impedire a un utente di interrompere un programma oppure di eseguire il ricaricamento del sistema operativo del computer. Dopo che un tasto è stato intercettato, non è possibile utilizzare l'istruzione INPUT o INKEY$ per ottenere il valore del tasto che ha provocato l'evento.

Errori

Se il valore del numero del tasto specificato non si trova nell'intervallo da 0 a 25 e non è 30 o 31, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita".

Suggerimenti

La gestione degli eventi rallenta l'esecuzione dei programmi, quindi è meglio non utilizzarla, a meno che non sia proprio indispensabile.

Esempio

Il programma di seguito riportato sfrutta la gestione degli eventi della tastiera per visualizzare le informazioni di aiuto, quando l'utente preme il tasto F1. Il programma inoltre intercetta il tasto STAMP quando viene premuto da solo o in combinazione con MAIUSC, con BLOCNUM o con entrambi. Quando viene premuta una qualsiasi di questa combinazione di tasti, il programma esegue la subroutine NoStampaSchermo.

```
CLS
LOCATE , , 1

ON KEY(1) GOSUB MostraAiuto                    'tasto funzione F1
KEY(1) ON

KEY 15, CHR$(&H80) + CHR$(&H37)                'tasto STAMP
ON KEY(15) GOSUB NoStampaSchermo
KEY(15) ON

KEY 16, CHR$(&H83) + CHR$(&H37)                'combinazione MAIUSC+STAMP
ON KEY(16) GOSUB NoStampaSchermo
KEY(16) ON
```

```
KEY 17, CHR$(&HA0) + CHR$(&H37)                    'tasto STAMP con attivo BLOC
NUM
ON KEY(17) GOSUB NoStampaSchermo
KEY(17) ON

KEY 18, CHR$(&HA3) + CHR$(&H37)                    'MAIUSC+STAMP con attivo BLOC
NUM
ON KEY(18) GOSUB NoStampaSchermo
KEY(18) ON

PRINT "Immettere il nome di un nuovo paziente: "
nome$ = ""
DO
  tast$ = INKEY$
  IF tast$ = CHR$(8) THEN       'controlla la pressione del tasto
                                'Backspace(Arretramento)
     LOCATE CSRLIN, POS(0) - 1
     PRINT " ";
     LOCATE CSRLIN, POS(0) - 1
     nome$ = LEFT$(nome$, LEN(nome$) - 1)
  ELSE
     PRINT tast$;
     nome$ = nome$ + tast$
  END IF
LOOP UNTIL tast$ = CHR$(13)    'esce dal ciclo quando l'utente preme Invio
PRINT "Hai inserito il nome: "; nome$
END

MostraAiuto:
     LOCATE 5, 1
     PRINT "Inserire il nome di un nuovo paziente nella seguente forma:"
     PRINT
     PRINT "   Cognome, Nome"
     LOCATE 2, 1
     RETURN

NoStampaSchermo:
     LOCATE 5, 1
     PRINT "Mi dispiace, ma non puoi stampare questa schermata!"
     BEEP
     LOCATE 2, 1
     RETURN
```

Capitolo 15

Stampanti

INTRODUZIONE

QBasic fornisce un gruppo di istruzioni che consentono di stampare e gestire l'output, sfruttando tutte le potenzialità offerte dal programma. Ovviamente, è possibile aprire la stampante come un file e utilizzare l'istruzione PRINT, come nel seguente esempio:

```
OPEN "LPT1:" AS #1
PRINT #1, "Il piccolo Principe"
```

Tuttavia l'istruzione LPRINT risulta più intuitiva e vantaggiosa, come illustrato nel seguente esempio:

```
LPRINT "Il piccolo Principe"
```

Si noti che aprendo un file corrispondente a una stampante e quindi utilizzando le istruzioni LPRINT e LPRINT USING, l'output risultante non è prevedibile.

Nella Tabella 15.1 sono riportate in ordine alfabetico le istruzioni e le funzioni relative alla stampante.

STAMPA DI UNA PARTE DI TESTO

Dato che l'operazione eseguita dalle istruzioni LPRINT e LPRINT USING, che dirigono l'output alla stampante, è analoga a quella effettuata da PRINT e PRINT USING, che dirigono l'output allo schermo è possibile trovare ulteriori informazioni consultando queste voci nel Capitolo 8. Questo capitolo si occupa soltanto delle questioni relative alla stampante.

Utilizzando LPRINT senza alcun parametro, si indica alla stampante di saltare una riga. Quando invece questa istruzione viene utilizzata con una variabile stringa o

Tabella 15.1 Le istruzioni e le funzioni di QBasic relative alla stampante

Istruzione o funzione	Descrizione
LPOS	Restituisce la posizione attuale della colonna di output all'interno del buffer della stampante.
LPRINT	Stampa l'output sulla stampante collegata alla porta parallela LPT1:.
LPRINT USING	Stampa una stringa formattata o un altro valore sulla stampante. Una stringa speciale indica il formato.
WIDTH LPRINT	Imposta la larghezza della colonna dell'output della stam pante.

numerica oppure con una costante stringa, ad esempio "Il piccolo Principe", la stampante stampa il valore e quindi un ritorno a capo e un avanzamento riga (CR/LF), che sposta la testina di stampa all'inizio della nuova riga. Se non si desidera che venga effettuata quest'ultima operazione è sufficiente immettere un punto e virgola alla fine dell'istruzione LPRINT; ad esempio l'istruzione:

```
LPRINT "Nessuna nuova riga.";
```

elimina il codice CR/LF, in modo che l'output successivo venga stampato sulla stessa riga. QBasic non inserisce spazi tra le voci, a meno che non vengano esplicitamente indicati; ad esempio, l'istruzione:

```
LPRINT Nome$; " "; Cognome$.
```

inserisce uno spazio tra il nome e il cognome. Le voci possono essere spaziate in base a intervalli regolari, separandole con una virgola in un'istruzione LPRINT. Ogni nuova voce viene stampata all'inizio della successiva zona di stampa composta da 14 spazi; l'istruzione:

```
LPRINT "Testo", "Ampiamente", "Spaziato"
```

produce l'output:

```
Testo         Ampiamente    Spaziato
```

Inserendo una virgola alla fine dell'istruzione LPRINT, è possibile spostare la testina di stampa in corrispondenza della successiva zona di stampa ed eliminare il codice CR/ LF.

L'istruzione LPRINT USING indica la stringa di formato e la voce o le voci da stampare. Ad esempio, l'istruzione di seguito riportata stampa *****$29.99.

```
LPRINT USING "**$####.##", 29.99
```

CONTROLLO DELLA STAMPANTE

L'istruzione LPRINT risulta utile anche quando è necessario inviare alla stampante speciali sequenze di controllo che consentono di accedere alle caratteristiche speciali della propria stampante.

QBasic presuppone che la stampante sia di tipo riga per riga. Con quasi tutte le stampanti (comprese le stampanti di pagina e laser) l'istruzione LPRINT CHR$(12) stampa un carattere di avanzamento pagina, facendo quindi avanzare la testina di stampa nella parte superiore della pagina successiva. La maggior parte delle stampanti ha una lunghezza di default per la pagina di 66 righe.

Si noti che l'istruzione LPRINT aggiunge un ritorno a capo (carattere ASCII 13) e uno di avanzamento riga (carattere ASCII 10) a una riga di output. Se questa istruzione viene utilizzata con una stampante impostata per inserire automaticamente propri codici di avanzamento riga, l'output contiene righe vuote in eccesso. Questo problema può essere risolto utilizzando il dispositivo di stampa DOS in modo binario, come nel seguente esempio.

```
OPEN "LPT1:" FOR BINARY AS #1
```

Quindi, per inviare l'output alla stampante è sufficiente utilizzare l'istruzione PUT, come riportato nella seguente parte di programma:

```
parole$ = "Il piccolo Principe"
PUT #1. parole$
```

L'istruzione PUT non inserisce né un ritorno a capo, né un codice di avanzamento riga.

L'istruzione WIDTH LPRINT imposta la larghezza della riga di stampa. Quando l'output raggiunge la larghezza specificata, la stampante stampa un codice CR/LF.

Se occorre controllare in modo più preciso la posizione della testina di stampa, è sufficiente utilizzare la funzione LPOS per determinarne la posizione nella riga in uso; a questo punto il programma può decidere se è necessario iniziare una nuova riga. LPOS può anche essere utilizzata per formattare l'output su una pagina: è sufficiente richiamare questa funzione e quindi spostare la testina di stampa in base a un numero specifico di spazi, che dipende dal valore restituito.

Bibliografia

Schneider, David L. *Handbook of BASIC.* 3d ed. New York, N.Y.: Brady, 1988.

LPOS

Si consulti anche

POS

Scopo

La funzione LPOS restituisce la posizione della colonna di output attuale della stampante alla porta specificata. Questa informazione può essere utilizzata dal programma per determinare quando iniziare una nuova riga di output, "portare a capo" le parole e così via.

Sintassi

```
LPOS(porta)
```

porta deve essere il numero 1, 2 o 3, che fa riferimento alla porta LPT1: (o PRN:), LPT2: o LPT3:.

Utilizzo

```
LPRINT nome$
col = LPOS(1)
```

Invia alla stampante la stringa contenuta in nome$ e quindi memorizza la colonna di output attuale di LPT1: nella variabile col.

Descrizione

Nella maggior parte dei casi i caratteri inviati alla stampante non vengono stampati immediatamente, ma vengono memorizzati nel buffer della stampante fino a quando il computer non trasmette una riga completa o un codice di avanzamento riga (LPRINT e PRINT inviano un ritorno a capo, a meno che l'istruzione termini con una virgola o un punto e virgola). Subito dopo l'avanzamento riga una chiamata a LPOS restituisce 1, a indicare che la stampante sta iniziando una nuova riga di output.

La funzione LPOS non restituisce in effetti la posizione della testina di stampa, ma il numero di caratteri che sono stati stampati sulla riga in uso.

Avvertenze

La funzione LPOS considera un carattere di tabulazione come un carattere singolo; tuttavia la stampante generalmente lo stampa come se fosse costituito da diversi spazi; quindi il valore restituito da LPOS non corrisponde alla posizione effettiva della testina di stampa, dopo che è stato stampato un carattere di tabulazione.

Spegnendo la stampante, la posizione della testina viene ripristinata in corrispondenza del margine sinistro e qualsiasi carattere presente nel buffer di stampa viene cancellato. Quando si riaccende nuovamente la stampante, QBasic non tiene conto del fatto che il buffer è stato cancellato e quindi il valore di LPOS è identico a quello presente prima che la stampante venisse spenta.

Esempio

Il programma di seguito riportato richiede di indicare il numero di colonna del margine sinistro e di quello destro e di digitare alcune parole da tastiera e in seguito stampa la riga di testo immessa, una parola alla volta, utilizzando i margini specificati. Prima di stampare una parola, ne verifica la lunghezza in base alla differenza tra la posizione del margine destro e il valore di LPOS; se la parola richiede spazio oltre il margine il programma la stampa all'inizio della riga successiva.

Il sottoprogramma SuccParola suddivide ogni riga di input in parole, una parola alla volta, e restituisce la parola (parola$) e la parte restante della stringa. Si noti che questo sottoprogramma viene definito con la parola chiave STATIC dato che deve mantenere i valori delle proprie variabili tra le chiamate.

```
DECLARE SUB SuccParola (unaStringa$, sep$, parola$)

sep$ = " "                              'usa uno spazio per separare le parole
INPUT "Margine colonna sinistro"; margineSin
INPUT "Margine colonna destro"; margineDes
PRINT "Inserire il testo una riga per volta."
PRINT "Premi <Invio> da solo per uscire dal programma."
GOSUB NuovaLinea                        'inizia la stampa su una nuova riga
DO
  LINE INPUT linea$
  IF LEN(linea$) = 0 THEN
     LPRINT                                       'LPRINT svuota il buffer
     LPRINT CHR$(12)
     SYSTEM
  END IF
  GOSUB Giustifica
LOOP
END

Giustifica:
     DO
        CALL SuccParola(linea$, sep$, parola$)  'ottiene la parola successiva
        IF (LEN(parola$) > (margineDes - LPOS(1))) THEN
           GOSUB NuovaLinea   'va alla riga successiva se la parola
                              'oltrepassa il margine destro
        END IF
        LPRINT parola$; " ";
     LOOP UNTIL LEN(parola$) = 0
     RETURN

NuovaLinea:
     LPRINT                             'va alla riga successiva
     FOR spazio% = 1 TO margineSin
         LPRINT " ";              'stampa degli spazi fino al margine sinistro
     NEXT spazio%
     RETURN


SUB SuccParola (unaStringa$, sep$, parola$) STATIC
    parola$ = ""
    strlungh% = LEN(unaStringa$)
    IF unaStringa$ = "" THEN EXIT SUB

    FOR car1% = 1 TO strlungh%
        IF INSTR(sep$, MID$(unaStringa$, car1%, 1)) = 0 THEN EXIT FOR
    NEXT car1%

    FOR car2% = 1 TO strlungh%
        IF INSTR(sep$, MID$(unaStringa$, car2%, 1)) = 0 THEN EXIT FOR
    NEXT car2%

    FOR car3% = 1 TO car2%
        IF INSTR(sep$, MID$(unaStringa$, car3%, 1)) = 0 THEN EXIT FOR
    NEXT car3%

    IF car1% > strlungh% THEN
       unaStringa$ = ""
       EXIT SUB
    END IF
```

```
    IF car2% > strlungh% THEN
       parola$ = MID$(unaStringa$, car1%)
       unaStringa$ = ""
       EXIT SUB
    END IF

    parola$ = MID$(unaStringa$, car1%, car2% - car1%)

    IF car3% > strlungh% THEN
       unaStringa$ = ""
    ELSE
       unaStringa$ = MID$(unaStringa$, car3%)
    END IF
END SUB
```

LPRINT

Si consulti anche

LPRINT USING, PRINT, PRINT USING

Scopo

L'istruzione LPRINT stampa il contenuto delle variabili numeriche o stringa oppure dei letterali sulla stampante. QBasic presuppone che la stampante sia collegata alla porta parallela (LPT1: o PRN:). Questa istruzione può anche essere utilizzata per inviare alla stampante codici di controllo.

Sintassi

```
LPRINT [dati[{; |,}dati...]] [; |,]
```

dati può essere una qualsiasi costante stringa, espressione o variabile. Se viene specificata più di una voce, è necessario separarle con virgole o punti e virgola.

Se tra due voci viene utilizzato un punto e virgola, la seconda viene stampata immediatamente dopo la prima; se invece viene utilizzata una virgola, la seconda voce viene stampata nella successiva zona di stampa (ogni zona ha una lunghezza di 14 caratteri). Un punto e virgola o una virgola alla fine dell'istruzione PRINT impediscono alla testina di stampa di spostarsi sulla riga successiva, dopo la stampa.

Utilizzo

```
LPRINT "Ciao";
```

Stampa la parola “Ciao” ma non sposta la testina di stampa sulla riga successiva.

```
LPRINT sottototale + tassa
```

Stampa la somma delle variabili sottototale e tassa.

```
LPRINT CHR$(12)
```

Invia alla stampante un carattere di avanzamento pagina.

Descrizione

L'istruzione LPRINT funziona esattamente come l'istruzione PRINT, tranne per il fatto che l'output viene inviato alla stampante parallela, invece che allo schermo. LPRINT può inoltre utilizzare la funzione CHR$ per inviare qualsiasi carattere alla stampante, compresi i comandi di controllo, in modo da poter accedere alle funzioni speciali della stessa.

Ogni tipo di stampante dispone di una serie diversa di comandi di controllo (per ulteriori informazioni, si consulti il manuale della stampante).

Errori

Se la stampante non è collegata o non è in linea QBasic visualizza il messaggio di errore: "Timeout di periferica".

Suggerimenti

QBasic non offre alcuna istruzione analoga a LPRINT per stampare direttamente su una stampante seriale, tuttavia a questo scopo è possibile sfruttare il comando del DOS:

```
MODE LPT1= COM1
```

per re-indirizzare l'output della stampante parallela alla porta seriale collegata a COM1; in questo caso, l'output di LPRINT e LPRINT USING viene inviato alla stampante seriale. Oppure, è possibile aprire il dispositivo COM1:, assegnargli un numero di file e quindi stampare utilizzando l'istruzione PRINT $NO.

Esempio

Il programma di seguito riportato utilizza LPRINT per impostare una stampante HP LaserJet e stampare un file su disco con un orientamento orizzontale e tipo carattere compresso. Questo formato potrebbe essere utile per la stampa dei listati di programma, a cui è necessario apportare diverse note.

```
DECLARE SUB Intestazione ()
CONST lineePerPagina% = 20          'righe per pagina inclusa l'intestazione
CONST VERO = 1
CONST FALSO = NOT VERO
primaVolta% = VERO        'flag utilizzato per evitare l'avanzamento pagina
                          'sulla prima pagina
'imposta la stampante
LPRINT CHR$(27); "&l1O"; 'imposta l'orientamento orizzontale
LPRINT CHR$(27); "(s0T"; 'imposta il carattere di stampa della linea
LPRINT CHR$(27); "&k2S"; 'imposta il formato compresso
LPRINT CHR$(27); "&l3d"; 'imposta la spaziatura doppia
LPRINT CHR$(27); "&l2E"; 'imposta un margine superiore di 2 linee

INPUT "Nome del file da stampare: "; file$   'specifica il file da stampare
ON ERROR GOTO ErrFile
OPEN file$ FOR INPUT AS #1                    'apre il file specificato

Intestazione                'stampa l'intestazione per la prima pagina
primaVolta% = FALSO  'rende Falso il flag perchè siano stampati i fine pagina
DO                                  'stampa una pagina
   LINE INPUT #1, linea$            'ottiene la riga successiva del file
   LPRINT linea$                    'stampa la riga
   currLinea% = currLinea% + 1      'conteggia le righe stampate
```

```
   IF currLinee% = lineePerPagina% THEN Intestazione
LOOP WHILE NOT EOF(1)

CLOSE #1                        'chiude il file di input
LPRINT CHR$(27); "E";           'reimposta la stampante per il lavoro successivo
END

ErrFile:                                  'routine di gestione degli errori
   errNum% = ERR
   SELECT CASE errNum%
   CASE 29
        PRINT "Errore di scrittura."
   CASE 30
        PRINT "Errore di lettura."
   CASE 52
        PRINT "Nome file errato."
   CASE 53
        PRINT "File non trovato."
   CASE 68
        PRINT "Dispositivo non disponibile. Assicurarsi che ";
        PRINT "la stampante sia collegata e on line."
   END SELECT
   SYSTEM                        'in caso di errore ritorna al sistema operativo
END

SUB Intestazione STATIC
    SHARED primaVolta%, currLinea%
    IF primaVolta% = FALSO THEN
       LPRINT CHR$(12)                          'avanzamento pagina
    ELSE
       numPagina% = 1
    END IF
    LPRINT UCASE$(file$); SPC(10);              'stampa il nome del file
    LPRINT DATE$; SPC(10);                      'stampa la data attuale
    LPRINT TIME$; SPC(5);                       'stampa l'ora attuale
    LPRINT SPC(5);
    LPRINT "Pagina "; numPagina%                'stampa il numero di pagina
    numPagina% = numPagina% + 1                 'incrementa il numero pagina
    currLinea% = 1                              'imposta la riga corrente
END SUB
```

LPRINT USING

Si consulti anche

LPRINT, PRINT USING

Scopo

L'istruzione LPRINT USING stampa numeri o stringhe formattati e indica la lunghezza e il formato di ogni voce stampata.

Sintassi

```
LPRINT USING stringaformato; var[; var...] [,|;]
```

stringaformato è una costante stringa o una variabile stringa, che può contenere una parte di testo e uno o più caratteri speciali di formattazione, che determinano in che modo vengono stampate le variabili.

var è un'espressione numerica o di tipo stringa che viene stampata nel formato indicato da stringaformato.

Inserendo una virgola alla fine dell'istruzione, la testina di stampa rimane sulla riga attuale e si sposta nella successiva zona di stampa, mentre il punto e virgola fa in modo che la testina di stampa rimanga posizionata sullo spazio direttamente successivo all'ultimo carattere stampato.

Utilizzo

```
LPRINT USING "##.#"; 24.99
```

Stampa il numero 25.0. Ogni segno di numero(#) opera da segnaposto per una cifra numerica. Dato che **stringaformato** indica soltanto tre cifre, **LPRINT USING** arrotonda il numero.

```
LPRINT USING "&"; "QBasic"
```

Stampa la parola QBasic.

Descrizione

L'istruzione **LPRINT USING** richiede una stringa speciale di formattazione che si trova in variabile stringa o è racchiusa tra doppi apici. La stringa di formattazione, che fornisce una maggiore versatilità nella formattazione dei numeri e delle stringhe, può essere utilizzata per stampare i numeri arrotondati a un determinato numero di posizioni decimali, preceduti dai segni di dollaro o da altri caratteri speciali, oppure stampati in notazione scientifica. Questa stringa può anche essere sfruttata per stampare le stringhe in formati diversi, ad esempio troncati o giustificati.

In modo analogo a **PRINT**, se alla fine della voce da stampare viene inserito un punto e virgola, vengono eliminati il ritorno a capo e l'avanzamento riga; se invece viene utilizzata una virgola, l'istruzione **LPRINT** che segue viene stampata nella successiva zona di stampa di 14 caratteri.

Errori

Se la stampante non è collegata oppure non è in linea, QBasic visualizza il messaggio di errore: "Timeout di periferica".

Esempio

Il programma di seguito riportato consente all'utente di immettere i numeri e le stringhe di formattazione, in modo da verificarne gli effetti; può essere utilizzato per esercitarsi nell'utilizzo dei formati di stampa. Per sfruttarlo con le stringhe, invece che con i valori numerici, è sufficiente modificare la variabile **num#** in una variabile stringa (ad esempio **unaStringa$**).

```
CLS
DO
  INPUT "Numero da stampare (0 per terminare) "; num#
  IF num# = 0 THEN EXIT DO
  INPUT "Formato da usare "; formato$
  IF formato$ = "" THEN
     LPRINT "Devi immettere una stringa di formato."
  ELSE
     LPRINT USING formato$; num#
  END IF
LOOP
```

WIDTH LPRINT

Si consulti anche

LPOS, POS, WIDTH (I/O di File), WIDTH (Schermo)

Scopo

L'istruzione WIDTH LPRINT indica la larghezza di un riga stampata. QBasic ne inizia una nuova, ogniqualvolta la testina di stampa raggiunge la fine di una riga.

Sintassi

```
WIDTH LPRINT colonna
```

colonna è un intero nell'intervallo da 1 a 255, che indica il numero massimo di colonne in ogni riga stampata.

Utilizzo

```
WIDT LPRINT 40
```

Stampa tutto l'output in non più di 40 colonne per riga.

Descrizione

L'istruzione WIDTH LPRINT indica la colonna in corrispondenza della quale la stampante inserisce un codice di ritorno a capo e di avanzamento riga (CR/LF). Qualsiasi altro codice di questo tipo inviato da un'istruzione LPRINT sposta la testina di stampa sulla riga successiva. Dopo aver richiamato l'istruzione WIDTH LPRINT, una riga non può superare la larghezza specificata.

Se la larghezza impostata è maggiore rispetto a quella della carta utilizzata dalla stampante, vengono troncati i caratteri specificati nell'istruzione LPRINT o LPRINT USING che si trovano tra il bordo della carta e la larghezza indicata, prima di stampare un codice CR/LF; specificando una larghezza di 255 colonne, questo codice viene eliminato. Anche se l'output di un'istruzione LPRINT o LPRINT USING supera le 255 colonne, la testina di stampa non si sposta sulla riga successiva, ma compie questa operazione soltanto quando il programma richiama una delle due istruzione sopra citate.

Esempio

Il programma di seguito riportato richiede all'utente di impostare la larghezza di riga per la stampante e di digitare le righe che il programma invia alla stampante.

```
INPUT "Immettere la larghezza della riga della stampante: "; colonna%
WIDTH LPRINT colonna%
PRINT "Immettere una riga per volta. Premi Invio per terminare."

DO
  LINE INPUT Linea$                         'ottiene la riga
  IF LEN(Linea$) = 0 THEN EXIT DO
  LPRINT Linea$;                            'non forza un fine riga
LOOP
LPRINT CHR$(12)                             'stampa un carattere di fine pagina
```

Capitolo 16

Porte di comunicazione

INTRODUZIONE

Le comunicazioni dei personal computer comportano il trasferimento dei dati tra un sistema e un altro; i dati possono essere file di programma (binari), tabelle elettroniche e record di database, documenti di testo, posta elettronica, immagini, o qualsiasi altra informazione che può essere codificata in un formato digitale. L'altro computer può essere un personal oppure un mainframe lontano mille miglia e utilizzare il DOS, l'OS/2, lo UNIX, o qualsiasi altro sistema operativo. I dati possono venire trasmessi mediante un cavo direttamente collegato a un altro computer oppure inviati attraverso le linee telefoniche con un modem.

I programmatori che utilizzano QBasic non devono necessariamente conoscere tutti i dettagli riguardanti i sistemi di trasmissione e di ricezione. Tutto ciò che è indispensabile sapere è che le comunicazioni di PC consentono di trasferire i dati verso e da la porta seriale di comunicazione, che è un collegamento fisico al PC e al suo buffer di memoria. Per quanto riguarda QBasic, la comunicazione si attua con la porta seriale, e non con un altro computer o un altro sistema operativo.

In questo capitolo viene illustrato l'utilizzo delle istruzioni di QBasic che consentono di aprire il collegamento della porta seriale, di definire un buffer temporaneo di memoria che contiene i dati, di segnalare quando i dati sono stati ricevuti e di chiudere il collegamento quando non è più necessario. Nella Tabella 16.1 sono riportate le istruzioni di QBasic relative alle comunicazioni.

Porte e buffer

Una *porta* di un PC consiste di tre elementi: un collegamento fisico (elettronico), una particolare posizione in memoria per le informazioni di stato relative al collegamento e un buffer di memoria per la memorizzazione temporanea dei dati in entrata e in uscita. QBasic può trasmettere o ricevere i dati mediante le porte seriali standard del

Tabella 16.1 Le istruzioni relative alle comunicazioni

Istruzione	Descrizione
COM ON/OFF/STOP	Attiva, disattiva o sospende la gestione degli eventi relativa alle comunicazioni.
ON COM GOSUB	Indica la subroutine a cui passa il controllo, quando una determinata porta di comunicazione riceve i dati.
OPEN COM	Apre un collegamento della porta seriale, assegnandogli un buffer e un numero di file. L'istruzione consente inoltre di indicare le caratteristiche di comunicazione, quali la velocità di trasmissione, il numero dei bit di dati, il numero dei bit di stop e il controllo del modem.

PC. È necessario tenere presente che QBasic supporta soltanto le porte seriali COM1 e COM2.

QBasic assegna un'area di memoria, definita buffer, a ogni porta; quest'area di memorizzazione temporanea è necessaria, dato che generalmente un programma deve eseguire anche altri compiti durante il trasferimento dei caratteri da e verso la porta di comunicazione. Se il programma dovesse interrompersi ogniqualvolta un carattere arriva alla porta, l'esecuzione progredirebbe lentamente, mentre con una velocità di trasmissione superiore, i caratteri andrebbero persi ancor prima di poter essere utilizzati. Per poter meglio gestire i dati, il buffer di input accumula i caratteri fino a quando diventa pieno (o quasi pieno); quindi, il programma trasferisce il contenuto del buffer sullo schermo, in un file o in un array di tipo stringa, in base all'applicazione utilizzata. In modo analogo, i caratteri in uscita vengono accumulati nel buffer di output e quindi inviati, quando il buffer è pieno.

IMPOSTAZIONE DEL COLLEGAMENTO

Quando vengono inviate informazioni attraverso QBasic, la prima fase consiste nel richiamare l'istruzione OPEN COM per indicare la porta seriale da utilizzare e lo stato del collegamento. Ad esempio, l'istruzione:

```
OPEN "COM1:1200,N,8,1" FOR RANDOM AS #1
```

indica che viene utilizzata la porta COM1: e che la velocità di trasmissione dei caratteri è di 1200 baud (circa 1200 bit per secondo). Quest'ultimo valore e le altre caratteristiche del collegamento devono essere identiche sia per il computer che riceve, che per quello che invia i dati; ciò significa che, se si desidera che il programma QBasic venga collegato a un sistema che consente collegamenti a una velocità di 300 baud, 1200 baud o 2400 baud, è necessario scegliere uno di questi valori ma non, ad esempio, 9600 baud.

L'istruzione riportata a titolo d'esempio utilizza gli argomenti N, 8, e 1 per indicare parità nessuna, 8 bit per carattere e 1 bit di stop. Il controllo della parità è un metodo che consente di individuare gli errori di trasmissione, garantendo che il numero di bit in un carattere con valore 1 (o 0, in base al tipo di controllo di parità) sia pari (o dispari); questo tipo di controllo viene eseguito impostando un bit di parità, il valore del quale indica se il numero di bit deve essere pari o dispari. Se il carattere giunge a destinazione senza il numero corretto di bit pari o dispari, il ricevente richiede che il carattere venga nuovamente inviato. I bit di dati sono i bit che contengono i dati in fase di trasmissione (ad esempio i caratteri ASCII). Si noti che 7 bit è un numero sufficiente per trasmettere uno qualsiasi dei 128 caratteri ASCII, mentre 8 bit sono necessari per i dati numerici o per i caratteri del set di caratteri estesi IBM. I bit di stop indicano la fine di una serie di dati di bit. Generalmente, il valore utilizzato con maggiore frequenza è 1 bit di stop; il valore di 2 bit è richiesto soltanto per le velocità di trasmissione più basse (110 baud o inferiore), mentre l'impostazione di 1.5 bit deve essere utilizzata soltanto quando vengono specificati 5 bit di dati.

La clausola FOR deve essere utilizzata in un'istruzione OPEN COM per indicare in che modo il programma utilizza i dati inviati e ricevuti dalla porta. FOR RANDOM, il valore di default in QBasic, indica che i dati vengono considerati come record in un file ad accesso casuale; la lunghezza del record può essere specificata mediante l'istruzione LEN. La modalità ad accesso casuale può risultare utile nel caso di invio di blocchi di dati a lunghezza fissa, ad esempio quando vengono implementati i protocolli di correzione degli errori, quale XMODEM, che protegge dagli errori di trasmissione. È anche possibile specificare le istruzioni FOR INPUT e OPEN COM in modo che il programma riceva soltanto l'input sequenziale, oppure FOR OUTPUT per l'invio solo dell'output sequenziale.

La clausola AS è la parte finale dell'istruzione d'esempio. AS #1 indica che è possibile accedere alla porta COM1: come file #1. Si noti che, dato che QBasic considera la porta di comunicazione come un file, è possibile aprirla e quindi ricevere l'input o inviare l'output al buffer corrispondente. QBasic utilizza buffer separati di trasmissione (TB) e di ricezione (RB); in QBasic la dimensione di default di ogni buffer è di 512 byte.

Altre impostazioni di comunicazione e controllo del modem

L'istruzione OPEN COM consente inoltre di modificare altre impostazioni di comunicazione: il modo in cui vengono considerati i dati (come caratteri ASCII o valori binari), le dimensioni dei buffer di invio e di ricezione, la stampa dei codici di avanzamento riga su una stampante seriale e l'intervallo tra la verifica dei segnali del modem e il verificarsi del *time-out.* Un time-out è il termine della comunicazione che si verifica quando viene superato il tempo concesso per definire il collegamento o eseguire l'operazione. Senza i time-out, il programma si interromperebbe nel caso di problemi di collegamento. Se non viene specificato alcun valore di time-out, QBasic utilizza quello di default, ovvero 1 secondo. In caso di problemi con il time-out, dovuti a collegamenti poco potenti o a diversità tra l'hardware che riceve e quello che invia, è possibile provare ad aumentare i valori di questo parametro.

CARICAMENTO E INVIO DEI CARATTERI

Per caricare i caratteri dal buffer di input, è sufficiente utilizzare innanzitutto la funzione EOF per determinare se vi sono caratteri da elaborare (dato che QBasic considera la periferica di comunicazione come un file, lo stato di "fine file" indica che non vi sono più caratteri da caricare). Quindi, è necessario utilizzare la funzione INPUT$ con quella LOC per caricare tutti i caratteri nel buffer. Le istruzioni INPUT o LINE INPUT non devono essere utilizzate, dato che potrebbero non caricare tutti i caratteri: INPUT si interrompe in corrispondenza di una virgola, di una nuova riga, e talvolta di uno spazio o di una coppia di doppi apici; LINE INPUT si interrompe in corrispondenza di una nuova riga. L'istruzione GET invece non deve essere utilizzata, poiché attende che il buffer venga riempito e quindi i dati potrebbero superare il limite ancora prima di essere trasferiti.

Il ciclo di seguito riportato illustra la tecnica consigliata per caricare i caratteri: attende fino a quando vi sono caratteri nella porta COM1: (partendo dal presupposto che la porta venga aperta correttamente) e quindi li visualizza sullo schermo.

```
DO
   IF NOT EOF(1) THEN    'la porta è stata aperta come file #1
      PRINT INPUT$(LOC(1), #1)
   END IF
LOOP
```

Per inviare i caratteri alla porta seriale, e quindi al sistema collegato, è possibile utilizzare qualsiasi istruzione PRINT o PUT corretta. Se ogni riga di input viene assegnata a una stringa denominata riga$, l'istruzione:

```
PRINT #1, riga$
```

memorizza i caratteri presenti in riga$ nel buffer di trasmissione, il cui contenuto viene poi inviato alla porta.

Dato che una sessione di comunicazione utilizza i file per l'input e l'output seriale, è sufficiente utilizzare alla fine del programma l'istruzione CLOSE per chiuderli.

GESTIONE DELLE COMUNICAZIONI CON LA GESTIONE DEGLI EVENTI

In molti casi, non si desidera che il programma inizi ad attendere dati dal buffer di comunicazione, fino a quando non viene attivata la comunicazione. QBasic presenta una serie di istruzioni che passano il controllo alla subroutine specificata, quando i dati vengono ricevuti. L'esecuzione di una subroutine al momento del verificarsi di un evento viene denominata gestione degli eventi.

L'istruzione ON COM GOSUB indica la subroutine che viene eseguita quando i caratteri sono disponibili per l'elaborazione nel buffer di comunicazione. Il programma deve anche eseguire un'istruzione COM ON per attivare la gestione degli eventi di comunicazione. L'istruzione COM STOP sospende l'esecuzione di questa funzione, ovvero i caratteri continuano ad arrivare al buffer, ma il controllo non passa alla

subroutine. Questa istruzione deve essere utilizzata per disattivare la gestione degli eventi, quando il programma esegue una funzione in cui il tempo è un elemento fondamentale. Tuttavia, è necessario assicurarsi di attivare nuovamente la gestione prima che il buffer sia troppo pieno, dato che in questo caso si perdono i dati e il programma potrebbe interrompersi. Quando si utilizza un'altra istruzione COM ON per ripristinare la gestione degli eventi, QBasic è in grado di ricordare qualsiasi evento che si è verificato mentre la gestione degli eventi di comunicazione era interrotta, e di eseguire la subroutine corretta, permettendo quindi al programma di mettersi alla pari con l'attività di comunicazione. L'istruzione COM OFF disattiva semplicemente la gestione degli eventi di comunicazione e non è in grado di ricordare alcun evento.

Nella sintassi delle istruzioni relative a questa caratteristica, è sempre necessario inserire tra parentesi, dopo la parola chiave COM, il numero della porta di comunicazione (1 o 2). Questo numero fa riferimento a COM1: o a COM2: e non deve essere confuso con il numero di file assegnato nell'istruzione OPEN COM.

Un programma che contiene la gestione degli eventi di comunicazione utilizza lo schema di seguito riportato (in questo esempio la porta è COM1:).

```
OPEN "COM1:1200,N,8,1" FOR RANDOM AS #1          'apre la connessione
ON COM(1) GOSUB Connessione                      'specifica la subroutine
COM(1) ON                                        'attiva la gestione degli
eventi

DO
    .
    .
    .
    COM(1) STOP                                  'inizia attività temporalmente critica
    .
    .
    .
    COM(1) ON                                    'riattiva la gestione degli eventi
LOOP
CLOSE
END

Connessione:                                     'subroutine di gestione degli eventi
    .
    .
    .
    RETURN
```

ALTRE INFORMAZIONI

Per eseguire comunicazioni seriali non è indispensabile avere un modem, né tantomeno due. Se si dispone di due computer che utilizzano il DOS, è possibile collegarne le porte con un cavo cosidetto "null modem" ed eseguire contemporaneamente i programmi QBasic su due macchine. Questo metodo consente di verificare l'input da un computer e l'output da un altro. Il collegamento può essere controllato sfruttando i programmi di esempio riportati nella voce ON COM GOSUB.

Può risultare difficile sviluppare programmi di comunicazione sofisticati, che devono gestire problemi relativi alla tempificazione, ai protocolli di trasmissione dei file, all'emulazione di terminale e così via. Per ampliare la propria conoscenza su questa materia, è possibile consultare i testi elencati nella bibliografia.

Bibliografia

Barkakati, Nabajyoti. "Programming the Serial Port with C," in *MS-DOS Papers*, by The Waite Group. Indianapolis, Ind.: Howard W. Sams & Co., 1988.

The Waite Group. *MS-DOS Developer's Guide.* 2d ed. Indianapolis, Ind.: Howard W. Sams & Co., 1989.

Willen, David C., and Jeffrey I. Kranz. *8088 Assembler Language Programming: The IBM PC.* 2d ed. Indianapolis, Ind.: Howard W. Sams & Co., 1984.

COM ON/OFF/STOP

Si consulti anche

ON COM GOSUB, OPEN COM

Scopo

Le istruzioni COM attivano, disattivano, o sospendono la gestione degli eventi di comunicazione. Questa caratteristica deve essere utilizzata, se si desidera che il programma esegua una subroutine ogniqualvolta la porta di comunicazione specificata riceve i caratteri.

Sintassi

```
COM(porta) ON
COM(porta) OFF
COM(porta) STOP
```

porta è il numero (1 o 2) della porta COM seriale che viene utilizzata.

Utilizzo

```
COM(1) ON
```

Attiva la gestione degli eventi di comunicazione alla porta COM1:.

Descrizione

QBasic consente di intercettare l'arrivo dei caratteri alla porta di comunicazione (seriale). L'istruzione COM ON deve essere utilizzata per attivare la gestione degli eventi di comunicazione in modo che, quando un carattere arriva, il controllo passi alla subroutine creata per gestire l'input (questa subroutine viene specificata nell'istruzione ON COM GOSUB).

La porta seriale desiderata deve essere aperta utilizzando l'istruzione OPEN COM, prima di essere in grado di intercettare un evento di comunicazione.

Se il programma sta eseguendo altre attività, in cui il tempo è un elemento fondamentale (ad esempio la gestione di un'altra periferica), potrebbe essere necessario sfruttare l'istruzione COM OFF per disattivare la gestione degli eventi in corrispondenza della porta specificata. I caratteri in arrivo continuano ad essere inseriti nel buffer di ricezione, ma se quest'ultimo arriva a contenere un numero di dati superiore alle proprie capacità, i dati vengono perse e il programma potrebbe essere interrotto.

L'istruzione COM STOP deve essere utilizzata per sospendere la gestione degli eventi di comunicazione da una porta specificata. I dati in arrivo continuano a essere inseriti nel buffer di input, anche se la subroutine specificata nell'istruzione ON COM GOSUB non viene eseguita, fino a quando non si utilizza nuovamente un'altra istruzione COM ON. QBasic imposta un flag che registra il verificarsi di un evento, in modo che l'intercettazione possa essere gestita dopo l'esecuzione della successiva istruzione COM ON.

Il programma esegue un'istruzione COM STOP implicita ogniqualvolta viene eseguita la subroutine specificata da ON COM GOSUB. Questo garantisce che la subroutine non venga eseguita in modo ricorsivo, quando si verifica un'altra occorrenza di un evento. Quando il programma individua un'istruzione RETURN e ritorna al modulo principale, imposta nuovamente la gestione degli eventi di comunicazione eseguendo un'istruzione COM ON implicita.

Avvertenze

Quando vengono sviluppati programmi e sottoprogrammi di comunicazione, è necessario utilizzare l'istruzione ON ERROR GOTO per intercettare qualsiasi errore che possa verificarsi. Questi programmi devono sempre essere sottoposti a test in base a diverse condizioni (velocità differenti, diversi gradi di disturbo sulla linea e così via), in modo da garantirne l'affidabilità.

È inoltre necessario assicurarsi che il buffer sia sufficientemente ampio da poter contenere i caratteri che potrebbero pervenire quando la gestione degli eventi è sospesa o interrotta. Si noti che potrebbe essere necessario aumentare le dimensioni del buffer, in caso di velocità di comunicazione più alte (2400 baud o superiore). Le dimensioni del buffer possono essere impostate utilizzando i parametri TB e RB nell'istruzione OPEN COM.

Esempio

Il programma di seguito riportato utilizza l'istruzione OPEN COM per attivare la gestione degli eventi di comunicazione e COM STOP per sospenderla fino a quando non viene premuto un tasto. All'inizio, il programma esegue la subroutine Evento ogniqualvolta i caratteri arrivano alla porta COM1:. Tuttavia, premendo un tasto questa funzione viene sospesa e il programma si interrompe visualizzando il messaggio: "Intercettato un evento sulla porta COM1:". Premendo un altro tasto, la gestione degli eventi riprende. Per uscire dal ciclo, è sufficiente premere ESC.

```
COM(1) ON                       'abilita la gestione della porta COM1:

ON COM(1) GOSUB Evento  'specifica la subroutine per la gestione degli eventi
OPEN "COM1:1200,N,8,1" FOR INPUT AS 2           'apre la porta

interruttore = -2                               'inizia con l'interruttore spento
DO
  PRINT "Aspetta che qualcosa accada ..."
  tasto$ = INKEY$
  IF tasto$ <> "" AND tasto$ <> CHR$(27) THEN
     interruttore = NOT interruttore  'inverte la posizione dell'interruttore
  END IF
  IF interruttore THEN
     PRINT "Sospensione della gestione degli eventi."
```

```
      COM(1) STOP
   END IF
   IF NOT interruttore THEN
      COM(1) ON
      PRINT "Ripresa della gestione degli eventi."
   END IF
LOOP UNTIL tasto$ = CHR$(27)        'esce dal ciclo quando l'utente preme Esc
CLOSE
END

Evento:
      PRINT "Intercettato un evento sulla porta COM1."
      RETURN
```

ON COM GOSUB

Si consulti anche

COM ON/OFF/STOP, OPEN COM

Scopo

L'istruzione ON COM GOSUB indica la subroutine a cui passa l'esecuzione, ogniqualvolta i dati vengono ricevuti nella porta di comunicazione specificata. Questa caratteristica consente al programma di eseguire altre operazioni senza per questo dover verificare costantemente l'attività di comunicazione.

Sintassi

```
ON COM(porta) GOSUB {riga | etichetta}
```

porta è il numero (1 o 2) della porta COM seriale che viene utilizzata.
riga o etichetta indicano la riga in cui inizia la subroutine a cui passa l'esecuzione.

Utilizzo

```
ON COM(1) GOSUB EseguiCom
```

Indica che viene eseguita la subroutine EseguiCom, quando i dati arrivano alla porta COM1:.

Descrizione

L'istruzione ON COM GOSUB specifica la subroutine di gestione degli eventi relativa alla porta seriale di comunicazione. Questa caratteristica rappresenta la capacità del programma di rispondere automaticamente ai segnali provenienti dall'hardware. In questo caso, i segnali indicano che sono stati ricevuti alcuni dati alla porta seriale di comunicazione.

Per impostare la gestione degli eventi, è sufficiente seguire le fasi di seguito riportate.

- Innanzitutto, occorre aprire la porta di comunicazione utilizzando l'istruzione OPEN COM.

- Quindi, è necessario specificare, mediante l'istruzione **ON COM GOSUB**, la porta e la subroutine che deve gestire gli eventi di comunicazione.
- Poi, occorre attivare la gestione degli eventi con un'istruzione **COM ON**, che indichi la porta da utilizzare.
- Infine, è sufficiente definire la subroutine che elabora il processo di comunicazione.

I programmi d'esempio di seguito riportati illustrano le modalità di utilizzo di queste fasi con la porta seriale COM1:. Si noti che le istruzioni di comunicazione di QBasic supportano soltanto due porte seriali, 1 si riferisce alla porta seriale COM1: e 2 a quella COM2:.

Esempio

In questa parte sono illustrati due programmi: il primo apre la porta COM1: per l'output e invia la stringa "Questo è un test...", fino a quando non viene premuto un tasto; il secondo apre la porta COM1: per ricevere l'input, specifica che la subroutine GestCom deve essere eseguita ogniqualvolta i dati arrivano alla porta e attiva la gestione degli eventi di comunicazione. La subroutine utilizza la funzione **INPUT$** per caricare qualsiasi carattere in attesa (la funzione **LOC** restituisce il numero di caratteri presenti nel buffer). Se si dispone di due computer, è possibile collegare un cavo di tipo null modem alle porte seriali, eseguendo il primo programma su un computer e il secondo su un altro. Se invece i due computer utilizzati sono collegati da un modem, per definire un collegamento è necessario che uno dei due componga il numero telefonico dell'altro. Ad esempio, dopo aver aperto la porta di comunicazione come file #1, l'istruzione:

```
PRINT #1, "ATDP1234567"
```

indica al modem Hayes compatibile di comporre il numero 123-4567.

Nelle istruzioni **OPEN COM** utilizzate in questi programmi, i parametri **CD0**, **CS0**, **DS0**, **OP0**, e **RS** impediscono i time-out e attendono l'individuazione delle *richieste di emissione* (RTS). Se questi parametri non vengono utilizzati, (oppure vengono specificati valori di time-out più grandi), è probabile che uno dei programmi immetta un time-out, mentre attende che l'altro invii o riceva i caratteri.

```
'apre la porta
OPEN "COM1:1200,N,8,1,CD0,CS0,DS0,OP0,RS" FOR OUTPUT AS #1

DO
  PRINT #1, "Questo è un test ..."
LOOP WHILE INKEY$ = ""
CLOSE                                   'chiude la porta
END
```

```
'apre la porta
OPEN "COM1:1200,N,8,1,CD0,CS0,DS0,OP0,RS" FOR INPUT AS #1
ON COM(1) GOSUB GestCom     'specifica la subroutine di gestione degli eventi
COM(1) ON                                    'attiva la gestione

CLS
DO
  LOCATE 1, 1
```

```
  PRINT "Attende dati sulla porta COM1."
  PRINT "Premi un tasto qualsiasi per terminare."
  IF INKEY$ <> "" THEN SYSTEM
LOOP
CLOSE                                       'chiude la porta
END

GestCom:
  DO WHILE NOT EOF(1)
     PRINT INPUT$(LOC(1), #1);              'visualizza i dati
  LOOP
  RETURN
```

OPEN COM

Si consulti anche

COM ON/OFF/STOP, ON COM GOSUB

Scopo

L'istruzione OPEN COM stabilisce un collegamento seriale di comunicazioni con un programma QBasic. È possibile impostare diverse caratteristiche del collegamento, quali la velocità di trasmissione (baud-rate), la parità, il numero di bit di dati e i bit di stop, e i valori di time-out per le linee modem.

Sintassi

```
OPEN "COMporta: elencoopz1 elencoopz2" [FOR modalità] AS [#]numfile [LEN=lunrec]
```

porta è la porta di comunicazione che si desidera aprire (1 = COM1:, 2 = COM2:).

elencoopz1 è un elenco dei quattro parametri di trasmissione illustrati nella Tabella 16.2. È necessario inserire una virgola come segnaposto per qualsiasi parametro non utilizzato.

elencoopz2 è un elenco dei parametri riportati nella Tabella 16.3. Questi parametri possono essere specificati in qualsiasi ordine, a patto che siano separati da una virgola.

modalità è la parola chiave INPUT, OUTPUT, o RANDOM.

numfile è il numero del file dal quale è possibile accedere alla porta di comunicazione.

lunrec è un intero che indica le dimensioni del record utilizzato con la porta. Questo parametro può essere utilizzato se il valore di modalità è RANDOM.

Utilizzo

```
OPEN "COM2:2400,E,7,1,BIN,CD0,CS0,DS0,OP0" FOR RANDOM AS #2
```

Apre un collegamento della porta seriale e fornisce le specifiche relative alla comunicazione dei dati. COM2: viene aperta in modo che abbia una velocità di trasmissione di 2400 baud, esegua un controllo di parità pari e utilizzi 7 bit di dati e 1 bit di stop. La porta non espande le tabulazioni, né inserisce un ritorno a capo; dato che non utilizza alcun time-out, deve attendere le trasmissione e la ricezione. Questa porta viene aperta come file ad accesso casuale, a cui è possibile accedere con #2.

Tabella 16.2 I primi quattro parametri dell'istruzione OPEN COM

Parametro	Significato	Commenti
Velocità	Velocità di trasmissione in baud (bit per secondo)	Utilizza il valore 75, 110, 150, 300, 600, 1200, 2400, 4800 o 9600. La maggior parte dei collegamenti telefonici ha una velocità di 1200 o 2400 baud. Il modem utilizzato deve supportare la velocità specificata.
Parità	Il tipo di controllo di parità utilizzato per l'individuazione degli errori	Utilizza N (nessuno), E (pari), O (dispari), S (spazio), M (contrassegno) o PE (attivazione controllo errori). Se non si sa quale utilizzare, occorre provare innanzitutto N.
Bit di dati	Numero di bit in ogni gruppo che rappresenta i dati effettivi	Utilizza i valori 5, 6, 7 o 8. Quest'ultimo valore deve essere utilizzato soltanto se non viene eseguito alcun controllo di parità; in caso contrario, si utilizzi 7.
Bit di stop	Numero dei bit utilizzati per contrassegnare la fine di ogni parola	Utilizza 1, 1.5 o 2. La scelta corretta è di solito 1. 2 deve essere utilizzato per velocità di trasmissione basse e 1.5 soltanto se il numero di bit di dati è 5.

Descrizione

L'istruzione OPEN COM deve essere utilizzata prima che il programma possa accedere alla porta seriale di comunicazione. Nella sintassi è necessario inserire, dopo la parola chiave OPEN, il numero della porta da utilizzare (COM1: o COM2:) e i parametri desiderati, separati da virgole. È possibile specificare qualsiasi parametro riportato nelle Tabelle 16.2 e 16.3.

Nella prima tabella sono elencati i primi quattro parametri dell'elenco, che devono apparire nell'ordine illustrato; se ne viene omesso uno, è necessario inserire una virgola come segnaposto.

I parametri elencati nella seconda tabella, che possono essere immessi in qualsiasi ordine, devono essere utilizzati dopo aver impostato quelli della Tabella 16.2. Subito dopo l'elenco dei parametri, è necessario inserire una delle seguenti parole chiave:

RANDOM	Per l'input e l'output casuale
INPUT	Per l'input sequenziale
OUTPUT	Per l'output sequenziale

RANDOM, che è l'opzione generalmente più sfruttata, viene utilizzata da QBasic come modalità di default, se non ne viene specificata un'altra; può essere seguita dalla parola chiave LEN e dalle dimensioni in byte del record utilizzato (il valore di default è 128).

Tabella 16.3 I parametri restanti dell'istruzione OPEN COM

Parametro	**Significato**	**Commenti**
ASC	Indica i caratteri come ASCII, espande le tabulazioni, forza un ritorno a capo alla fine di ogni riga e considera CTRL+Z un contrassegno EOF.	L'impostazione standard per inviare e ricevere file di testo ASCII.
BIN	Non esegue nessuna delle azioni specificate da ASC.	Questa è l'impostazione di default. Si utilizzi ASC o BIN, ma non entrambi.
CD[m]	Imposta il tempo di timeout (in millisecondi) sulla riga Data Carrier Detect (DCD).	m deve essere un numero nell'intervallo da 0 a 65535; il valore 0, che è quello di default, disabilita il timeout.
CS[m]	Imposta il tempo di timeout in (millisecondi) sulla riga Pronto per Trasmettere (CTS).	m deve essere un numero nell'intervallo da 0 a 65535; il valore di default è 1000 (1 secondo).
DS[m]	Imposta il tempo di timeout (in millisecondi) sulla riga Insieme dei Dati Pronto (DS).	m deve essere un numero nell'intervallo da 0 a 65535; il valore di default è 1000 (1 secondo).
LF	Invia un carattere di avanzamento riga dopo Invio.	Questa impostazione viene utilizzata per le operazioni di stampa con una stampante seriale.
OP[m]	Indica quanto deve attendere (in millisecondi) OPEN COM prima che tutte le linee di comunicazione vengano aperte.	m deve essere un numero nell'intervallo da 0 a 65535; il valore di default è 1000 (1 secondo).
RB[n]	Imposta le dimensioni in byte del buffer di ricezione.	byte deve essere un numero nell'intervallo da 0 a 23767; il valore di default è 512.
RS	Disattiva la modalità di individuazione della Richiesta di Emissione (RTS).	
TB[n]	Imposta le dimensioni in byte del buffer di trasmissione.	byte deve essere un numero nell'inter vallo da 0 a 23767; il valore di default è 512.

Commenti

I problemi legati al tempo possono risultare difficili da gestire. Si inizi utilizzando le condizioni meno impegnative che l'applicazione è in grado di tollerare, specificando i parametri di time-out a 0 (nessun time-out), impostando buffer di ampie dimensioni e una bassa velocità (ad esempio 1200 baud, invece di 2400 o superiore). L'impostazione utilizzata con maggiore frequenza è: nessuna parità, 8 bit di dati e 1 bit di stop. Dopo aver acquisito maggiore dimestichezza con le problematiche legate alla comunicazione, si può iniziare a variare l'istruzione OPEN COM in modo che corrisponda alle esigenze del proprio programma.

Errori

Il messaggio di errore: "Nome del file errato", visualizzato durante la compilazione di un'istruzione OPEN COM, generalmente indica un errore di sintassi. Inoltre, possono verificarsi diversi altri errori durante l'operazione di comunicazione. Gli errori DOS che fanno riferimento a "errore di lettura" o "errore di scrittura" sulla periferica COM1: o COM2: indicano di solito che la porta non è stata aperta. Il messaggio di errore: "Timeout di periferica" indica che il programma non ha ricevuto le informazioni nel periodo di tempo specificato, mentre il messaggio: "errore interno" potrebbe indicare che il buffer è troppo pieno (di solito quello di ricezione) e quindi devono esserne aumentate le dimensioni.

Avvertenze

Si noti che il parametro ASC (testo ASCII) nell'istruzione OPEN COM non supporta la funzione di sincronizzazione XON/XOFF. I caratteri Ctrl+S e Ctrl+Q vengono considerati come caratteri normali, anziché segnali di inizio o interruzione della trasmissione dei dati.

Esempio

Il programma di seguito riportato richiede all'utente di immettere il numero della porta, la velocità di trasmissione, i bit di dati e i bit di stop, eseguendo quindi l'istruzione OPEN COM con questi valori.

```
CONST VERO = 1, FALSO = NOT VERO
CLS

DO
valido% = VERO
PRINT "Quale porta vuoi utilizzare"
INPUT "1) COM1: 2) COM2: 3) Uscita "; scelta%
SELECT CASE scelta%
    CASE 1
         porta$ = "COM1:"
    CASE 2
         porta$ = "COM2:"
    CASE 3
         SYSTEM
    CASE ELSE
         PRINT "Scegli 1, 2, o 3."
         valido% = FALSO
END SELECT
LOOP UNTIL valido% = VERO

PRINT
DO
valido% = VERO
PRINT "Quale velocità di trasmissione"
INPUT "1) 300 baud 2) 1200 baud 3) 2400 baud 4) 9600 baud "; scelta%
SELECT CASE scelta%
    CASE 1
         veloc$ = "300"
    CASE 2
         veloc$ = "1200"
    CASE 3
         veloc$ = "2400"
    CASE 4
```

```
          veloc$ = "9600"
     CASE ELSE
          PRINT "Scegli 1, 2, 3 o 4."
          valido% = FALSO
END SELECT
LOOP UNTIL valido% = VERO

PRINT
DO
valido% = VERO
PRINT "Quale parità vuoi utilizzare"
INPUT "1) Pari 2) Dispari 3) Nessuna "; scelta%
SELECT CASE scelta%
     CASE 1
          parit$ = "E"
     CASE 2
          parit$ = "O"
     CASE 3
          parit$ = "N"
     CASE ELSE
          PRINT "Scegli 1, 2, o 3."
          valido% = FALSO
END SELECT
LOOP UNTIL valido% = VERO

PRINT
DO
valido% = VERO
PRINT "Quanti bit di dati per byte"
INPUT "1) 5  2) 6  3) 7  4) 8 "; scelta%
SELECT CASE scelta%
     CASE 1
          bitDati$ = "5"
     CASE 2
          bitDati$ = "6"
     CASE 3
          bitDati$ = "7"
     CASE 4
          bitDati$ = "8"
          IF parit$ <> "N" THEN
             PRINT "Con 8 bit di dati devi utilizzare parità=N"
             PRINT "Imposta la parità a nulla (N)."
             parit$ = "N"
          END IF
     CASE ELSE
          PRINT "Scegli 5, 6, 7 o 8."
          valido% = FALSO
END SELECT
LOOP UNTIL valido% = VERO

PRINT
DO
valido% = VERO
PRINT "Quanti bit di stop"
INPUT "1) 1  2) 1.5  3) 2  "; scelta%
SELECT CASE scelta%
     CASE 1
          bitStop$ = "1"
     CASE 2
          bitStop$ = "1.5"
     CASE 3
          bitStop$ = "2"
     CASE ELSE
          PRINT "Scegli 1, 2, o 3."
          valido% = FALSO
END SELECT
```

```
LOOP UNTIL valido% = VERO

portaSpec$ = porta$ + veloc$ + "," + parit$ + "," + bitDati$ + "," + bitStop$
PRINT "Ecco le impostazioni delle comunicazioni: "; portaSpec$

'apre la porta di comunicazione
OPEN portaSpec$ FOR RANDOM AS #2
```

Capitolo 17

File

INTRODUZIONE

Le istruzioni di QBasic che gestiscono i file forniscono un collegamento fondamentale tra il programma e il mondo esterno. La maggior parte dei programmi applicativi sono utili perché i dati che l'utente immette possono essere salvati, ovvero una volta inseriti sono sempre disponibili su file. Memorizzando una lettera standardizzata, un foglio paga o un elenco delle scorte in magazzino, è poi possibile accedere immediatamente a informazioni che sono costate giorni o settimane di fatica.

Un file, memorizzato in qualsiasi modo, è semplicemente un insieme di byte; il metodo in base al quale questi dati vengono interpretati fornisce loro un significato. Generalmente, i programmatori raggruppano le informazioni utilizzando le seguenti strutture:

- *Campi.* Un campo è un pezzo singolo di informazione, analogo a una semplice variabile, che può contenere un nome, una data di nascita, il numero di una voce dell'inventario e così via (la *lunghezza del campo* viene generalmente misurata in byte).
- *Record.* Un record è un insieme di campi che costituiscono una serie di informazioni correlate ed è analogo a una variabile record definita da TYPE...END TYPE; può contenere un nome, un indirizzo, le informazioni relative a un dipendente (la *lunghezza del record* è la lunghezza di un singolo record e viene di solito misurata in byte).
- *File.* Un file è una serie di record, simile a un array di variabili record; può comprendere i record relativi a tutti i dipendenti di una società (la *lunghezza del file* è la lunghezza combinata di tutti i record presenti nel file e viene generalmente misurata in byte).

File sequenziali

Un *file sequenziale* è un flusso continuo di caratteri senza campi, né record. Dato che questo tipo di file è organizzato a riga, invece che a record, la posizione esatta delle informazioni è imprevedibile: ad esempio per individuare il numero di riga 100, è necessario innanzitutto esaminare le righe numerate da 1 a 99. I file sequenziali assomigliano a una cassetta musicale: per individuare una canzone che si trova a metà, è necessario far scorrere velocemente tutte le canzoni precedenti.

I file sequenziali possono contenere soltanto i caratteri ASCII o quelli del set di caratteri estesi IBM; ogni numero viene rappresentato da 1 byte per cifra.

File ad accesso casuale

Un *file ad accesso casuale* supporta completamente le strutture record. In effetti, questo tipo di file contiene una sequenza di record e un programma può leggere soltanto i record in esso contenuti oppure scriverne altri. Dato che tutti i record presenti in un file ad accesso casuale hanno la stessa lunghezza, QBasic è in grado di calcolare la posizione esatta di ognuno. Accedere a un record è semplice come individuare l'ultima canzone su un compact disk; dato che si conosce il punto esatto in cui ogni record (canzone) inizia e termina, è sufficiente passare alla posizione corretta.

File binari

Un *file binario* è una variazione sia dei file ad accesso casuale che di quelli sequenziali. Questo tipo di file può supportare campi e record, ma è in grado di operare anche senza di essi. Se si desidera ignorare i campi e le strutture dei record, generalmente presenti in un file ad accesso casuale, è possibile aprirlo come file binario. Dato che QBasic considera questo tipo di file come una sequenza di byte (non di caratteri, come in un file sequenziale), esso può essere utilizzato per contenere qualsiasi tipo di informazione.

QBasic fornisce le istruzioni e le funzioni che permettono di gestire in modo efficace i tipi diversi di file e le relative strutture. In questa parte vengono riepilogate le capacità di I/O dei file di QBasic e offerti alcuni suggerimenti relativi all'utilizzo. Nella Tabella 17.1 sono riportate in ordine alfabetico le istruzioni e le funzioni di I/O.

APERTURA DI UN FILE O DI UN DISPOSITIVO

QBasic fa riferimento ai file mediante un numero. L'istruzione OPEN associa un file a un numero corrispondente, in modo che si possa successivamente accedere al file indicando quel numero. Se non si conosce con sicurezza il numero corretto è sufficiente richiamare la funzione FREEFILE, che restituisce il numero del file successivo disponibile per l'apertura.

Tabella 17.1 Le istruzioni e le funzioni di I/O su file

Istruzione o funzione	**Descrizione**
CLOSE	Chiude un file o un dispositivo aperti.
CVD	Converte una stringa di 8 byte, restituita dalla funzione MKD$, nel corrispondente valore numerico a precisione doppia.
CVDMBF	converte una stringa di 8 byte contenente un valore nel formato binario Microsoft nel corrispondente valore numerico a precisione doppia nel formato IEEE.
CVI	Converte una stringa di 2 byte, restituita dalla funzione MKI$, in un valore numerico intero.
CVL	Converte una stringa di 4 byte, restituita dalla funzione MKL$, nel corrispondente valore numerico intero lungo.
CVS	Converte una stringa di 4 byte, restituita dalla funzione MKS$, nel corrispondente valore numerico a precisione semplice.
CVSMBF	Converte una stringa di 4 byte contenente un valore nel formato binario Microsoft nel corrispondente valore numerico a precisione semplice nel formato IEEE.
EOF	Restituisce -1 (vero), se il puntatore del file è posizionato alla fine di un file.
FIELD	Definisce la struttura dei record da utilizzare in un file ad accesso casuale.
FILEATTR	Restituisce le informazioni relative a un file.
FILES	Visualizza i nomi dei file che corrispondono alle specifiche fornite.
FREEFILE	Restituisce il primo numero di file non associato con un file aperto.
GET (I/O su file)	Legge i dati da un file binario o ad accesso casuale.
INPUT #	Legge i valori da un file sequenziale o da un dispositivo.
INPUT$ (I/O su file)	Legge un numero specificato di byte da tastiera, da un file o da un dispositivo.
LINE INPUT #	Legge i dati da un file sequenziale o da un dispositivo.
LOC	Restituisce la posizione attuale del puntatore in un file aperto.
LOCK	Limita o impedisce l'accesso ad un file condiviso in rete.
LOF	Restituisce il numero di byte in un file aperto.
LSET	Trasferisce il valore di un'espressione o di una variabile nel campo specificato del buffer di un file ad accesso casuale, allineandolo a sinistra.
LTRIM$	Restituisce una copia di un'espressione stringa senza spazi iniziali.
MKD$	Converte un numero a precisione doppia in una stringa di 8 byte.
MKDMBF$	Converte un valore in precisione doppia nel formato IEEE in una. stringa di 8 byte nel formato numerico binario Microsoft.
MKI$	Converte un numero intero in una stringa di 2 byte per l'output a un file ad accesso casuale.

Tabella 17.1 Le istruzioni e le funzioni di I/O su file (continua)

Istruzione o funzione	Descrizione
MKL$	Converte un numero intero lungo in una stringa di 4 byte per l'output a un file ad accesso casuale.
MKS$	Converte un numero a precisione semplice in una stringa di 4 byte per l'output a un file ad accesso casuale.
MKSMBF$	Converte un valore in precisione semplice nel formato IEEE in una stringa di 4 byte nel formato numerico binario Microsoft per l'output a un file ad accesso casuale.
NAME	Rinomina un file o una directory.
OPEN	Apre un file.
PRINT #	Scrive i dati non formattati in un file sequenziale.
PRINT # USING	Scrive i dati formattati in un file sequenziale.
PUT (I/O su file)	Scrive i dati in un file binario o in uno ad accesso casuale.
RESET	Chiude tutti i file.
RSET	Trasferisce il valore di un'espressione o di una variabile nel campo specificato del buffer di un file ad accesso casuale, allineandolo a destra.
RTRIM$	Restituisce una copia di un'espressione stringa senza spazi finali.
SEEK (Funzione)	Restituisce la posizione attuale in un file.
SEEK (Istruzione)	Si posiziona in un punto specificato del file.
UNLOCK	Sblocca un file in modo che gli altri programmi possano accedervi.
WIDTH (I/O su file)	Imposta la larghezza massima di un file sequenziale.
WRITE #	Scrive i dati in un file sequenziale.

L'istruzione OPEN esegue diverse altre operazioni: ad esempio crea un file se quello specificato non esiste, imposta le modalità di accesso, gestisce il buffer del file e l'accesso a quest'ultimo in un ambiente multiutente.

La modalità di accesso specificata in un'istruzione OPEN determina il modo in cui il programma creato può utilizzare il file. Per specificare tale modalità è sufficiente utilizzare una delle seguenti parole chiave: INPUT, OUTPUT, APPEND, RANDOM o BINARY. Le prime tre aprono il file sotto forma di file sequenziale che contiene una sequenza di caratteri; il programma può caricare i dati da questo tipo di file soltanto aprendolo con INPUT e può scrivere i dati in un file sequenziale solo aprendolo con OUTPUT o APPEND. La parola chiave OUTPUT indica che viene creato un nuovo file sequenziale, mentre APPEND specifica che viene aperto un file esistente e che i dati immessi vengono inseriti alla fine.

L'utilizzo della parola chiave RANDOM in un'istruzione OPEN consente di aprire un file nella modalità ad accesso casuale. OPEN utilizza quest'ultima modalità come default, se non viene specificata la modalità di accesso. La parola chiave BINARY deve invece essere utilizzata per aprire un file in modalità binaria.

Un buffer è un'area della memoria in cui un programma inserisce i dati che vengono scritti nel file o letti dal file. Dato che la lettura delle informazioni da disco è un processo meccanico (quindi lento), QBasic sfrutta il buffer per immagazzinare i dati, mentre il programma continua l'esecuzione. Per aumentare ulteriormente la velocità di esecuzione del programma, ogni volta che esso accede a un file su disco utilizzando un'operazione di lettura QBasic inserisce i dati nel buffer; quando il programma effettua un'altra operazione di lettura, non deve necessariamente accedere al disco, ma può ottenere i dati dal buffer.

I programmi che operano in rete possono condividere o limitare l'accesso ai file. La clausola LOCK nell'istruzione OPEN consente di specificare il tipo di accesso che gli altri programmi possono utilizzare per aprire i file. Nella Tabella 17.2 sono riportate le opzioni possibili.

In un ambiente di rete, il tipo di accesso desiderato può essere indicato inserendo la clausola ACCESS nell'istruzione OPEN. Le opzioni possibili sono: ACCESS READ (solo lettura), ACCESS WRITE (solo scrittura) e ACCESS READ WRITE (lettura e scrittura). Se un altro programma utilizza un'istruzione OPEN con una clausola LOCK per aprire un file e il proprio programma cerca di aprire lo stesso file con un'istruzione OPEN contenente una clausola ACCESS che indica un accesso analogo a quello specificato nella clausola LOCK, quest'ultimo non è in grado di aprire il file.

Tabella 17.2 Le definizioni della modalità di accesso in rete

Clausola	Descrizione
(Default)	Nessun'altra elaborazione può aprire il file.
SHARED	Qualsiasi elaborazione può aprire il file in qualsiasi modalità.
LOCK READ	Nessuna elaborazione può aprire il file con la clausola ACCESS READ; nessun'altra elaborazione può già avere i diritti ACCESS WRITE
LOCK READ WRITE	Nessuna elaborazione può aprire il file con la clausola ACCESS READ WRITE; nessun'altra elaborazione può già avere i diritti ACCESS READ WRITE

CHIUSURA DI UN FILE O DI UN DISPOSITIVO

L'istruzione OPEN consente di accedere a un file, mentre CLOSE termina la comunicazione con il file, cancella tutti i dati presenti nel corrispondente buffer e se necessario scrive il contenuto del buffer su disco. L'istruzione CLOSE può chiudere i file specificati oppure tutti i file contemporaneamente, mentre RESET permette di chiudere tutti i file e di cancellare i relativi buffer.

GESTIONE DEI FILE AD ACCESSO CASUALE

Un file ad accesso casuale è composto da una sequenza di record, che sono suddivisi in campi. L'istruzione TYPE...END TYPE consente di creare un tipo definito dall'utente che indica un numero di campi e quindi dichiarare le variabili record relative al tipo specificato. I file ad accesso casuale possono contenere record definiti in questo modo.

Dato che le versioni precedenti di Basic non supportano l'istruzione TYPE...END TYPE, è necessario utilizzare in questo caso l'istruzione FIELD per specificare i campi dei record (che devono essere stringhe) in un file ad accesso casuale. Per assegnare i valori ai campi, occorre invece sfruttare le istruzioni LSET e RSET, dato che FIELD alloca lo spazio per ogni campo nel buffer associato al file.

La lettera L in LSET sta ad indicare che la stringa viene giustificata a sinistra, mentre la R di RSET indica invece che la stringa viene giustificata a destra. LSET inserisce un valore a sinistra di un campo, riempiendo di spazi la parte destra, RSET invece immette un valore nella parte destra, aggiungendo gli spazi nella parte sinistra.

Dato che l'istruzione FIELD può definire soltanto campi composti da stringhe, Basic offre diverse funzioni che convertono i numeri in stringhe e altre funzioni corrispondenti che eseguono l'operazione contraria. QBasic supporta queste funzioni per garantire la compatibilità con le altre versioni di Basic. Nella Tabella 17.3 sono riportate le funzioni relative alla conversione di stringhe e numeri.

Le funzioni MKSMBF$ e MKDMBF$ convertono i numeri nel formato IEEE in stringhe contenenti numeri nel formato numerico binario Microsoft. Le funzioni CVSMBF e CVDMBF convertono le stringhe contenenti numeri nel formato numerico binario Microsoft in numeri nel formato IEEE. Queste funzioni consentono di accedere ai file creati dalle versioni precedenti di BASIC, che supportano soltanto il formato Microsoft.

Ad esempio, se il programma creato comprende il campo ImportoDovuto$, che è stato definito in un'istruzione FIELD, le istruzioni di seguito riportate potrebbero essere utilizzate per scrivere i valori numerici in un file su disco.

Tabella 17.3 Le funzioni relative alla conversione di stringhe e numeri

Tipo numerico	Da numero a stringa	Da stringa a numero
Intero	MKI$	C V I
Intero lungo	MKL$	C V L
A precisione semplice	MKS$	C V S
MBF* a precisione semplice	MKSMBF$	C V S M B F
A precisione doppia	MKD$	C V D
MBF* a precisione doppia	MKDMBF$	C V D M B F

* MBF sta per Microsoft Binary Format, ovvero il formato numerico utilizzato in GW-BASIC, BASICA e nella versione 3.0 e precedenti di QuickBASIC, le versioni successive di QuickBASIC e di QBASIC utilizzano il formato numerico IEEE.

```
LSET ImportoDovuto$ = MKD$(169.85)
PUT #1
```

La funzione MKD$ converte il numero 169.85 in una stringa, mentre LSET la inserisce nel campo ImportoDovuto$, che può quindi essere scritto su file. L'istruzione PUT scrive il record in uso nel file su disco.

Per leggere il valore di questo campo dal file, è possibile utilizzare le seguenti istruzioni:

```
GET #1
ImportoDovuto# = CVD(ImportoDovuto$)
```

L'istruzione GET carica un record da disco. ImportoDovuto# è una variabile a precisione doppia e la funzione CVD converte ImportoDovuto$ da una stringa nuovamente in un valore a precisione doppia.

LETTURA DEI DATI DAI FILE AD ACCESSO CASUALE O BINARI

L'istruzione GET viene utilizzata per leggere i record contenuti nei file ad accesso casuale o binari. In essa deve essere specificato sia il numero del record (il numero di byte in un file binario), che la variabile da leggere. Ad esempio, per leggere il primo record di un file ad accesso casuale, il cui numero è 3, e assegnarlo quindi alla variabile record info, è sufficiente utilizzare l'istruzione:

```
GET #3, 1, info
```

Anche la lettura dei dati dai file binari è altrettanto semplice. L'istruzione di seguito riportata inizia a leggere in corrispondenza del numero di byte 129 in un file binario il cui numero è 2 e quindi assegna il valore letto alla variabile fatture.

```
GET #2, 129, fatture
```

Quando viene definito un record utilizzando l'istruzione FIELD e poi questo record viene letto da un file ad accesso casuale, non è possibile specificare la variabile record a cui vengono assegnati i valori nell'istruzione GET. Quando non viene indicata alcuna variabile record, GET carica un record e lo inserisce nel buffer del file; a questo punto è necessario utilizzare le istruzioni INPUT#, LINE INPUT# o PRINT#, dopo GET, per leggere i valori dal buffer del file. L'utilizzo delle variabili record è più vantaggioso rispetto all'istruzione FIELD. Se la versione di Basic utilizzata supporta l'istruzione TYPE...END TYPE, essa deve essere sicuramente utilizzata nel caso dei file ad accesso casuale.

È possibile inoltre utilizzare la funzione INPUT$ per leggere i valori da un file ad accesso casuale o da un file binario. Questa funzione legge il numero specificato di caratteri presenti in un file. Il numero massimo di caratteri in file ad accesso casuale è rappresentato dalla lunghezza di un record, mentre in un file binario è di 32767 byte.

SCRITTURA DEI DATI NEI FILE AD ACCESSO CASUALE O BINARI

L'istruzione PUT deve essere utilizzata per scrivere i record nei file ad accesso casuale e binari. In essa deve essere specificato sia il numero del record (il numero di byte in un file binario), che la variabile da scrivere. Ad esempio, per scrivere i valori contenuti nella variabile record info nel record 31 del file ad accesso casuale il cui numero è 1, è sufficiente utilizzare l'istruzione:

```
PUT #1, 31, info
```

Anche l'operazione di scrittura in un file binario è semplice. L'istruzione di seguito riportata scrive il contenuto della variabile magazzino in un file binario, il cui numero è 12, partendo dal byte 1537.

```
PUT #12, 1537, magazzino
```

Quando si scrive un record definito in un'istruzione FIELD, la variabile non deve essere specificata nell'istruzione PUT. In questo caso, PUT scrive nel file qualsiasi valore inserito nel buffer dall'istruzione LSET o RSET.

LETTURA DEI DATI DAI FILE SEQUENZIALI

Per leggere i file sequenziali è possibile utilizzare le istruzioni INPUT # e LINE INPUT# oppure la funzione INPUT$. Generalmente, l'istruzione INPUT # viene utilizzata per leggere i numeri, mentre LINE INPUT # consente di leggere una riga intera di testo. La funzione INPUT$ invece legge il numero specificato di caratteri da un file sequenziale.

L'istruzione INPUT # legge i dati da un file sequenziale fino a quando non raggiunge un delimitatore, sia esso uno spazio, una virgola, un ritorno a capo o un avanzamento riga. Ad esempio, l'istruzione di seguito riportata utilizza INPUT # per leggere una variabile stringa e una variabile di tipo intero da un file il cui numero è 3.

```
INPUT #3, NomeStudente$, grado%
```

L'istruzione LINE INPUT # è più flessibile, dato che riconosce soltanto un delimitatore, ovvero l'associazione tra il ritorno a capo e l'avanzamento riga. Quindi LINE INPUT # è in grado di leggere una riga intera in una variabile, compreso qualsiasi delimitatore presente su quella riga, ma non di leggerne diverse; inoltre può assegnare i dati del file soltanto a una variabile stringa, non numerica. L'istruzione di seguito riportata, che legge una riga da un file il cui numero è 5 e la assegna alla variabile codice$, illustra l'utilizzo di LINE INPUT#.

```
LINE INPUT #5, codice$
```

La funzione INPUT$ legge un numero specificato di caratteri (al massimo 32767 e comunque non più di quelli presenti nel file) da un file sequenziale. Questa funzione, al contrario delle istruzioni precedenti, non riconosce alcun delimitatore, permettendo quindi di leggere qualsiasi carattere. L'istruzione di seguito riportata, che illustra

l'utilizzo di INPUT$, legge un carattere da un file il cui numero è 3 e lo assegna alla variabile carattere$.

```
carattere$ = INPUT$(1, #3)
```

SCRITTURA DEI DATI NEI FILE SEQUENZIALI

L'istruzione PRINT # è quella maggiormente utilizzata per scrivere su file sequenziali; funziona quasi come PRINT, che visualizza l'output sullo schermo. Ad esempio, la seguente istruzione scrive il numero 123.15 in un file il cui numero è 1:

```
PRINT #1, 123.15
```

L'istruzione WRITE # è analoga a PRINT #, tranne per il fatto che inserisce una virgola tra le espressioni nel file e racchiude le variabili stringa tra doppi apici; questo consente a INPUT # di leggere il file più facilmente. Ad esempio, l'istruzione:

```
WRITE #1, "BOSIO, GIORGIO", 012170, 150.25
```

scrive nel file corrispondente al numero 1 il seguente output:

```
"BOSIO, GIORGIO",012170,150.25
```

ELIMINAZIONE DEGLI SPAZI DALLE STRINGHE

Se una stringa comprende alcuni spazi a destra o a sinistra (come succede quando si utilizza l'istruzione LSET o RSET per i file ad accesso casuale), è possibile utilizzare la funzione LTRIM$ o RTRIM$ per eliminarli. Ad esempio, se viene utilizzata una variabile per il nome di una persona e un'altra per il cognome e si desidera creare lettere standardizzate, in cui il nome e il cognome appaiono insieme, è possibile eliminare gli spazi aggiuntivi posti a destra della stringa che contiene il nome. A questo scopo, è sufficiente utilizzare la seguente istruzione:

```
PRINT #assegni, "Pagare all'ordine di"; RTRIM$(Nome$); " "; Cognome$
```

Se si desidera eliminare sia gli spazi a destra che quelli a sinistra, è possibile utilizzare entrambe le funzioni, come riportato nel seguente esempio:

```
degenti$ = LTRIM$(RTRIM$(degenti$))
```

SPOSTAMENTO ALL'INTERNO DI UN FILE

Per ogni file aperto, QBasic mantiene un puntatore che indica la posizione al suo interno; qualsiasi operazione di input o di output eseguita dal programma inizia in quella posizione, a meno che non venga specificato diversamente nelle istruzioni utilizzate. Il puntatore del file può essere spostato in qualsiasi posizione, richiamando l'istruzione SEEK. Nel caso dei file ad accesso casuale, è necessario specificare il numero

del record a cui si desidera posizionare il puntatore; se invece questa istruzione viene utilizzata con un file sequenziale o binario, occorre indicare il numero del byte in cui deve essere posizionato il puntatore. Ad esempio per spostare il puntatore al record 256 di un file ad accesso casuale il cui numero è 17 è sufficiente utilizzare la seguente istruzione:

```
SEEK #17, 256
```

DETERMINAZIONE DELLE CARATTERISTICHE DEI FILE

Sebbene utilizzando QBasic non sia necessario conoscere tutti i minimi dettagli relativi ai file, è possibile in ogni caso accedere a tali informazioni. La funzione FILEATTR, ad esempio, restituisce la modalità (INPUT, OUTPUT, APPEND, RANDOM o BINARY) in base alla quale è stato aperto un file. Questa funzione può anche essere utilizzata per restituire il file handle utilizzato dal DOS per accedere al file. Probabilmente, è necessario conoscere questo valore soltanto se nei programmi QBasic si stanno utilizzando routine scritte in linguaggio assembler. La funzione EOF verifica se il puntatore si trova alla fine del file e può essere utilizzata in un ciclo che legge tutto il contenuto di un file, come riportato nel seguente esempio:

```
DEFINT A-Z

NumFile = FREEFILE
OPEN "REPORT.RPT" FOR INPUT AS #NumFile

DO UNTIL EOF(NumFile)
   LINE INPUT #NumFile, testo$
   LPRINT testo$
LOOP

CLOSE
```

La funzione LOF restituisce le dimensioni di un file in byte e può essere sfruttata per calcolare il numero di record presenti in un file ad accesso casuale, utilizzando la seguente formula:

```
numRecord = LOF(file) \ lunghezzaRecord 'divisione intera
```

QBasic fornisce inoltre le funzioni LOC e SEEK. La prima restituisce il numero di record scritti o letti per ultimi per i file ad accesso casuale, il numero di byte diviso per 128 per i file sequenziali, la posizione del byte letto o scritto per ultimo per i file binari. La seconda restituisce la posizione del record successivo che viene scritto o letto per i file ad accesso casuale, la posizione del byte successivo da leggere e scrivere per i file sequenziali o binari.

GESTIONE DEI FILE SU DISCO

QBasic comprende diverse funzioni che consentono di gestire i file.

L'istruzione FILES visualizza un elenco della directory in uso o di quella specificata; è analoga al comando DIR/W del DOS, che visualizza i nomi dei file suddivisi in

colonne orizzontalmente sullo schermo. FILES può essere utilizzata per consentire all'utente di verificare i file di dati disponibili.

L'istruzione NAME modifica il nome di un file esistente, proprio come il comando RENAME del DOS. Tuttavia, l'istruzione di QBasic è più potente rispetto a quella del DOS, dato che può essere sfruttata anche per spostare i file tra le directory su disco. Ad esempio, la seguente istruzione sposta un file nella directory RAPPORTO e gli assegna il nome RAPPORT1.RPT:

```
NAME "C:\ACCT\RAPPORTO.RPT" AS "C:\RAPPORT1\RAPPORT1.RPT"
```

BLOCCO DEI FILE

Quando si utilizza una rete locale, possono sorgere dei conflitti se diversi programmi cercano di accedere allo stesso file. La versione 3.1 del DOS e quelle successive supportano il blocco dei file e dei record: se un file o un record viene bloccato, soltanto un programma può accedervi; se un altro desidera accedere a un file o a un record bloccati, deve attendere fino a quando il programma li sblocca.

QBasic fornisce le istruzioni LOCK e UNLOCK per gestire questo tipo di operazione; la prima consente di bloccare tutto un file o soltanto una parte di esso, la seconda consente di sbloccarlo.

Si noti che Microsoft garantisce il supporto per QBasic soltanto nei sistemi LAN Manager di Microsoft, PC Network di IBM, MS-Network e altri sistemi compatibili. Anche se è possibile ottenere dei risultati anche utilizzando altre reti, ad esempio Novell e 3Com, possono sempre sorgere dei problemi. Se la maggior parte della programmazione ha a che fare con ambienti di rete diversi da quelli supportati da Microsoft, è bene prendere in considerazione l'utilizzo di un file manager prodotto da un'altra casa che li supporti.

IMPOSTAZIONE DELLA LARGHEZZA MASSIMA DEI FILE SEQUENZIALI

Se si creano file sequenziali che devono essere stampati su una stampante che utilizza un foglio di carta stretto, è meglio limitare la larghezza di ogni riga nel file, in modo che corrisponda ai margini del foglio. A questo scopo è sufficiente utilizzare l'istruzione WIDTH che, se utilizzata con un numero di file, consente di impostare la larghezza delle righe: quelle che superano la larghezza indicata vengono portate nella riga successiva, consentendo in questo modo di risparmiare il tempo necessario a calcolare la larghezza massima.

Bibliografia

Dettmann, Terry. *DOS Programmer's Reference.* Indianapolis, Ind.: Que, 1988.

Duncan, Ray, et al. *The MS-DOS Encyclopedia.* Redmond, Wash.: Microsoft Press, 1988.

Norton, Peter, and Richard Wilton. *The New Peter Norton Programmer's Guide to the IBM PC &PS/2.* Redmond, Wash.: Microsoft Press, 1988.

Prata, Stephen, with Harry Henderson. *The Waite Group's Microsoft QuickBASIC Primer Plus.* Redmond, Wash.: Microsoft Press, 1990.

The Waite Group. *MS-DOS Developer's Guide.* 2d ed. Indianapolis, Ind.: Howard Sams & Co., 1989.

The Waite Group. *Using PC DOS.* Indianapolis, Ind.: Howard Sams & Co., 1989.

CLOSE

Si consulti anche

OPEN

Scopo

L'istruzione CLOSE chiude uno o più file o dispositivi aperti e deve essere utilizzata dopo aver concluso tutte le operazioni di input e di output.

Sintassi

```
CLOSE [[#]numerofile[,[#]numerofile]...]
```

numerofile è il numero di un file o di un dispositivo aperti.

Utilizzo

```
CLOSE
```

Chiudi tutti i file e i dispositivi aperti.

```
CLOSE 5, #15
```

Chiude i file o i dispositivi che erano stati aperti con i numeri 5 e 15; il simbolo di numero è facoltativo.

Descrizione

L'istruzione CLOSE consente di chiudere un file precedentemente aperto con un'istruzione OPEN, che consente invece di allocare lo spazio del buffer per i record contenuti in un file. L'istruzione CLOSE scrive i dati restanti nello spazio del buffer nel file o nel dispositivo corretti e quindi libera lo spazio di memoria riservato al buffer. Se questa istruzione viene utilizzata senza parametri, vengono chiusi tutti i file e i dispositivi aperti.

Dopo aver chiuso un file, non è possibile farvi riferimento con un'altra istruzione di I/O fino a quando esso non viene specificato in un'altra istruzione OPEN. Quando un file viene chiuso, è possibile utilizzare il numero associato a quel file per riaprirlo o per aprire un altro file, ovvero qualsiasi file chiuso può essere riaperto con un numero di file non utilizzato.

Anche se, quando un programma esegue la propria ultima istruzione oppure un'istruzione CLEAR, END, RESET, RUN o SYSTEM, QBasic chiude tutti i file e i dispositivi, è bene inserire in ogni caso un'istruzione CLOSE alla fine del programma.

Commenti

La chiusura di un file che non è stato aperto non provoca alcun errore.

Avvertenze

Tutti i campi definiti da un'istruzione FIELD vengono impostati come stringhe nulle, dopo che il file è stato chiuso. Se si desidera invece salvare il contenuto dei campi, è necessario assegnarlo a delle variabili.

Esempio

Il programma di seguito riportato apre un file sequenziale e una stampante, quindi scrive la data e l'ora in entrambi e infine li chiude.

```
OPEN "RAPPORTO.TXT" FOR OUTPUT AS #5    'apre un file sequenziale per l'output
OPEN "LPT1:" FOR OUTPUT AS #6           'apre il dispositivo di stampa

PRINT #5, "La data di oggi è "; DATE$   'scrive la data sul file
PRINT #5, "e l'ora è "; TIME$           'scrive l'ora sul file

PRINT #6, "La data di oggi è "; DATE$   'scrive la data sul dispositivo di
                                        'stampa
PRINT #6, "e l'ora è "; TIME$           'scrive l'ora sul dispositivo di
                                        'stampa
PRINT #6, CHR$(12)                      'scrive un carattere di fine pagina
                                        'sul dispositivo di stampa

CLOSE #5, #6                            'chiude il file e il dispositivo
```

CVD

Si consulti anche

CVDMBF, DOUBLE, FIELD, MKD$, MKDMBF$

Scopo

La funzione CVD converte una stringa di 8 byte, restituita dalla funzione MKD$, nel corrispondente valore numerico a precisione doppia; può essere utilizzata dopo aver letto la rappresentazione sotto forma di stringa di un numero a precisione doppia in un file ad accesso casuale, che contiene i record definiti dall'istruzione FIELD.

Sintassi

```
CVD(stringa)
```

stringa è una variabile stringa di 8 byte creata dalla funzione MKD$.

Utilizzo

```
dovuto# = CVD(ImportoDovuto$)
```

Assegna alla variabile dovuto# il valore a precisione doppia contenuto nella variabile stringa ImportoDovuto$.

Descrizione

La funzione CVD converte una stringa di 8 byte nel corrispondente valore numerico a precisione doppia. La stringa che questa funzione converte deve essere contenuta in una variabile stringa creata dalla funzione MKD$.

Commenti

Se dalla riga di comando viene specificato il parametro /MBF, QBasic presuppone che il parametro della stringa per la funzione CVD sia un numero in formato binario Microsoft; in caso contrario, lo considera un numero a precisione doppia nel formato IEEE.

Esempio

Il programma di seguito riportato converte i valori in stringhe, scrivendole nel file ad accesso casuale SOL.DAT e quindi convertendo ogni stringa memorizzata nel file in un valore numerico. Il file contiene informazioni relative al sistema solare, compreso il nome del pianeta, il diametro, il *perielio* e l'*afelio*, la temperatura più alta e quella più bassa e l'anno in cui è stato scoperto (il perielio di un'orbita è la distanza minima del pianeta al sole; l'afelio è la distanza massima del pianeta al sole).

Il programma consente innanzitutto di scegliere il numero del record che si desidera leggere dal file; le funzioni CVL, CVD, CVDMBF, CVS, CVSMBF, e CVI vengono utilizzate per visualizzare i valori dei campi del record. Quindi il programma sfrutta le funzioni MKL$, MKD$, MKDMBF$, MKS$, MKSMBF$ e MKI$ per preparare i nuovi valori dei campi da scrivere nel file.

```
OPEN "SOL.DAT" FOR RANDOM AS #1 LEN = 45
FIELD #1, 15 AS nome$, 4 AS diametro$, 8 AS perielio$, 8 AS afelio$
FIELD #1, 35 AS fittizio$, 4 AS altaTemp$, 4 AS bassaTemp$, 2 AS
annoScoperta$

CLS
INPUT "Su quale pianeta si desiderano informazioni"; numPianeta%
GET #1, numPianeta%

PRINT "Il diametro del pianeta "; RTRIM$(nome$); " è ";
PRINT CVL(diametro$); " chilometri"
PRINT "Il perielio del pianeta "; RTRIM$(nome$); " è ";
PRINT CVD(perielio$); " chilometri"
PRINT "L'afelio del pianeta "; RTRIM$(nome$); " è ";
PRINT CVDMBF(afelio$); " chilometri"
PRINT "La temperatura massima del pianeta "; RTRIM$(nome$); " è ";
PRINT CVS(altaTemp$); " gradi"
PRINT "La temperatura minima del pianeta "; RTRIM$(nome$); " è ";
PRINT CVSMBF(bassaTemp$); " gradi"
PRINT "Il pianeta è stato scoperto nel "; CVI(annoScoperta$)
PRINT

INPUT "Vuoi modificare il record [S/N] ", risposta$
IF UCASE$(risposta$) = "S" THEN
   INPUT "Nome"; nuovoNome$
   INPUT "Diametro: ", nuovoDiametro&
   INPUT "Perielio: ", nuovoPerielio#
   INPUT "Afelio: ", nuovoAfelio#
```

```
    INPUT "Temperatura massima: ", nuovaaltaTemp!
    INPUT "Temperatura minima: ", nuovabassaTemp!
    INPUT "Anno della scoperta: ", nuovoannoScoperta%
    LSET nome$ = nuovoNome$
    LSET diametro$ = MKL$(nuovoDiametro&)
    LSET perielio$ = MKD$(nuovoPerielio#)
    LSET afelio$ = MKDMBF$(nuovoAfelio#)
    LSET altaTemp$ = MKS$(nuovaaltaTemp!)
    LSET bassaTemp$ = MKSMBF$(nuovabassaTemp!)
    LSET annoScoperta$ = MKI$(nuovoannoScoperta%)
    PUT #1, numPianeta%
END IF

CLOSE #1
```

CVDMBF

Si consulti anche

CVD, DOUBLE, FIELD, MKD$, MKDMBF$

Scopo

La funzione CVDMBF converte una stringa di 8 byte contenente un valore nel formato binario Microsoft nel corrispondente valore numerico a precisione doppia nel formato IEEE; può essere utilizzata dopo aver letto la rappresentazione sotto forma di stringa di un numero a precisione doppia in un file ad accesso casuale, che contiene le stringhe nel formato binario Microsoft; il file deve contenere record definiti dall'istruzione FIELD.

Sintassi

```
CVDMBF(stringa)
```

stringa è una variabile stringa di 8 byte creata dalla funzione MKDMBF$ o da MKD$ in una versione di Basic che supporta soltanto il formato binario Microsoft.

Utilizzo

```
dovuto# = CVDMBF(ImportoDovuto$)
```

Assegna alla variabile dovuto# il valore a precisione doppia contenuto nella variabile stringa ImportoDovuto$.

Descrizione

La funzione CVDMBF converte una stringa di 8 byte nel corrispondente valore numerico a precisione doppia, nel formato IEEE. La stringa che questa funzione converte deve essere contenuta in una variabile stringa creata dalla funzione MKD$, in una versione di Basic che supporta soltanto il formato binario Microsoft, o dalla funzione MKDMBF$.

Commenti

QBasic presuppone che il parametro della stringa nella funzione CVDMBF sia un numero in formato binario Microsoft.

Esempio

Si consulti l'esempio nella voce CVD.

CVI

Si consulti anche

FIELD, INTEGER, MKI$

Scopo

La funzione CVI converte una stringa di 2 byte, restituita dalla funzione MKI$, in un valore numerico intero; può essere utilizzata dopo aver letto la rappresentazione sotto forma di stringa di un numero intero in un file ad accesso casuale che contiene i record definiti dall'istruzione FIELD.

Sintassi

```
CVI(stringa)
```

stringa è una variabile stringa di 2 byte creata dalla funzione MKI$.

Utilizzo

```
conto% = CVI(NumConto$)
```

Assegna alla variabile conto% il valore intero contenuto nella variabile stringa NumConto$.

Descrizione

La funzione CVI converte una stringa di 2 byte in un valore numerico intero. La stringa che questa funzione converte deve essere contenuta in una variabile stringa o in un'espressione creata dalla funzione MKI$.

Esempio

Si consulti l'esempio nella voce CVD.

CVL

Si consulti anche

FIELD, LONG, MKL$

Scopo

La funzione CVL converte una stringa di 4 byte, restituita dalla funzione MKL$, nel corrispondente valore numerico intero lungo; può essere utilizzata dopo aver letto la rappresentazione sotto forma di stringa di un numero intero lungo in un file ad accesso casuale che contiene i record definiti dall'istruzione FIELD.

Sintassi

```
CVL(stringa)
```

stringa è una variabile stringa di 4 byte creata dalla funzione MKL$.

Utilizzo

```
serie& = CVL(NumSerie$)
```

Assegna alla variabile serie& il valore di tipo intero lungo contenuto nella variabile stringa NumSerie$.

Descrizione

La funzione CVL converte una stringa di 4 byte nel corrispondente valore numerico intero lungo. La stringa che questa funzione converte deve essere una variabile stringa creata dalla funzione MKL$.

Esempio

Si consulti l'esempio nella voce CVD.

CVS

Si consulti anche

CVSMBF, FIELD, MKS$, MKSMBF$

Scopo

La funzione CVS converte una stringa di 4 byte, restituita dalla funzione MKS$, nel corrispondente valore numerico a precisione semplice; può essere utilizzata dopo aver letto la rappresentazione sotto forma di stringa di un numero a precisione semplice in un file ad accesso casuale, che contiene i record definiti dall'istruzione FIELD.

Sintassi

```
CVS(stringa)
```

stringa è una variabile stringa di 4 byte creata dalla funzione MKS$.

Utilizzo

```
dovuto! = CVS(ImportoDovuto$)
```

Assegna alla variabile dovuto! il valore a precisione semplice contenuto nella variabile stringa ImportoDovuto$.

Descrizione

La funzione CVS converte una stringa di 4 byte nel corrispondente valore numerico a precisione semplice. La stringa che questa funzione converte deve essere contenuta in una variabile stringa creata dalla funzione MKS$.

Commenti

Se dalla riga di comando viene specificato il parametro /MBF, QBasic presuppone che il parametro della stringa per la funzione CVS sia un numero in formato binario Microsoft; in caso contrario, lo considera un numero a precisione semplice nel formato IEEE.

Esempio

Si consulti l'esempio nella voce CVD.

CVSMBF

Si consulti anche

CVD, DOUBLE, FIELD, MKD$, MKDMBF$

Scopo

La funzione CVSMBF converte una stringa di 4 byte contenente un valore nel formato binario Microsoft nel corrispondente valore numerico a precisione semplice nel formato IEEE; può essere utilizzata dopo aver letto la rappresentazione sotto forma di stringa di un numero a precisione semplice in un file ad accesso casuale, che contiene le stringhe nel formato binario Microsoft; il file deve contenere record definiti dall'istruzione FIELD.

Sintassi

```
CVSMBF(stringa)
```

stringa è una variabile stringa di 4 byte creata dalla funzione MKSMBF$ o da MKS$ in una versione di Basic che supporta soltanto il formato binario Microsoft.

Utilizzo

```
dovuto! = CVSMBF(ImportoDovuto$)
```

Assegna alla variabile dovuto! il valore a precisione semplice contenuto nella variabile stringa ImportoDovuto$.

Descrizione

La funzione CVSMBF converte una stringa di 4 byte nel corrispondente valore numerico a precisione semplice, nel formato IEEE. La stringa che questa funzione converte deve essere contenuta in una variabile stringa creata dalla funzione MKS$, in una versione di Basic che supporta soltanto il formato binario Microsoft, o dalla funzione MKSMBF$.

Commenti

QBasic presuppone che il parametro della stringa nella funzione CVSMBF sia un numero nel formato binario Microsoft.

Esempio

Si consulti l'esempio nella voce CVD.

EOF

Si consulti anche

LOC, LOF, SEEK

Scopo

La funzione EOF restituisce -1 (vero) se il puntatore del file è posizionato alla fine di un file, 0 (falso) se nel file sono contenute informazioni anche dopo il puntatore del file; può essere utilizzata per decidere se sia il caso di continuare a elaborare il file.

Sintassi

```
EOF(numerofile)
```

numerofile è il numero del file associato al file aperto.

Utilizzo

```
IF EOF(1) THEN GOTO FineLavoro
```

Passa l'esecuzione alla riga contenente l'etichetta FineLavoro, se il puntatore del file si trova alla fine del file associato al numero 1.

Descrizione

Durante l'elaborazione del contenuto di un file, è necessario sapere quando interrompersi perché si è raggiunta la fine del file. Per quanto riguarda i file sequenziali, EOF restituisce -1 (vero), quando l'istruzione di input più recente raggiunge la fine del file, mentre per quanto riguarda i file ad accesso casuale e quelli binari, EOF restituisce -1 (vero) soltanto se l'ultima istruzione GET non è stata in grado di leggere tutto il record; in questo caso nell'istruzione GET è stato specificato un numero di record che si trovava oltre la fine del file.

I dispositivi di QBasic SCRN: KYBD: CONS: e LPTn: non hanno una fine logica del file.

EOF opera in modo diverso, se il numero specificato è il file associato al dispositivo di comunicazione (COM1: o COM2:) e se il dispositivo COM2n era aperto in modalità ASCII o binaria. Quando si utilizza COM1: o COM2: nella modalità di comunicazione ASCII, EOF restituisce -1 (vero), soltanto dopo aver letto un carattere Ctrl+Z (codice ASCII 26); quindi, EOF continua a restituire -1, fino a quando il file non viene chiuso. Nella modalità di comuncazione binaria, la funzione EOF restituisce -1 quando la coda di input è vuota; in caso contrario, restituisce 0 (falso).

Errori

Al contrario della maggior parte delle istruzioni di I/O su file, EOF non accetta un segno di numero (#) con il numero del file. In questo caso, QBasic visualizza il messaggio di errore: "Era atteso: espressione".

Se la funzione EOF viene utilizzata con un numero di file che non è stato specificato da un'istruzione OPEN, QBasic visualizza il messaggio di errore: "Nome o numero di file errato".

Esempio

Il programma di seguito riportato utilizza EOF per integrare le righe di due file sequenziali in un terzo file. Il file risultante contiene (nell'ordine) la prima riga dal file #1, la prima riga dal file #2, la seconda riga dal file #1 e così via.

```
OPEN "Primo" FOR INPUT AS #1        'apre il primo file sorgente
OPEN "Secondo" FOR INPUT AS #2      'apre il secondo file sorgente
OPEN "Terzo" FOR INPUT AS #3        'apre il file destinazione

DO
  IF NOT EOF(1) THEN                'controlla per la fine del file
     LINE INPUT #1, Primo$          'legge una linea di testo
     PRINT #3, Primo$               'scrive la linea sul file destinazione
  END IF

  IF NOT EOF(2) THEN                'controlla per la fine del file
     LINE INPUT #2, Secondo$        'legge una linea di testo
     PRINT #3, Secondo$             'scrive la linea sul file destinazione
  END IF
LOOP UNTIL EOF(1) AND EOF(2)        'ripete fino a quando entrambi i file sono
                                    'stati letti

CLOSE 1, 2, 3
```

FIELD

Si consulti anche

CLOSE, GET (I/O), LSET, OPEN, PUT, RSET

Scopo

L'istruzione FIELD definisce la struttura dei record da utilizzare in un file ad accesso casuale. Se possibile, tuttavia, è meglio utilizzare la struttura TYPE...END TYPE per eseguire questa operazione.

Sintassi

```
FIELD [#]numerofile, largcampo AS var [,largcampo AS var] ...
```

numerofile è il numero di un file aperto; può essere preceduto dal segno di numero (#).

largcampo è il numero, in byte, dei caratteri contenuti nel campo.

variabilestringa è un campo che contiene la stringa da scrivere o da leggere dal file.

Utilizzo

```
OPEN "CLIENTI.DAT" FOR RANDOM AS #1 LEN =200
FIELD #1, 32 AS NomeCliente$, 32 AS IndirCliente$, 32 AS CodPostCitCliente$
```

Apre il file CLIENTI.DAT e definisce un record che contiene tre campi, ognuno dei quali contiene una stringa di 32 caratteri.

Descrizione

Ogni record in un file ad accesso casuale può comprendere molti campi diversi, che sono posizioni a cui è possibile accedere mediante un nome. I record definiti dall'istruzione FIELD sono analoghi alle strutture indicate dall'istruzione TYPE...END TYPE.

È possibile specificare un singolo numero di file in diverse istruzioni FIELD, consentendo a variabili diverse di condividere contemporaneamente gli stessi dati in un record. Dato che l'istruzione FIELD definisce la struttura del record per un file, l'utilizzo di più di un'istruzione FIELD consente di impostare un record la cui definizione sarebbe troppo lunga per una sola riga. Si consideri il seguente esempio:

```
FIELD #1, 10 AS Nome$, 20 AS Cognome$
FIELD #1, 30 AS fittizio$, 6 AS DataNascita$      '30= somma larghezza riga
                                                  'precedente
FIELD #1, 36 AS fittizio$, 32 AS Indirizzo$       '36= somma larghezza riga
                                                  'precedente
```

Quando queste istruzioni indicano fittizio$ come un campo, assegnano alla variabile il numero di byte utilizzati nelle precedenti istruzioni FIELD. Queste istruzioni eseguono lo stesso compito della seguente singola istruzione:

```
FIELD #1, 10 AS Nome$, 20 AS Cognome$, 6 AS DataNascita$, 32 AS Indirizzo$
```

Quando i campi vengono definiti in un'istruzione FIELD, QBasic li posiziona all'interno dello spazio di memoria del buffer. Dato che i campi non sono variabili comuni di QBasic, è necessario utilizzarli con una certa precauzione. Dato che QBasic libera lo spazio del buffer quando il file viene chiuso, i campi definiti con FIELD diventano non definiti, dopo che il file è stato chiuso; inoltre, questo tipo di campi non può essere utilizzato nelle istruzioni INPUT, LINE INPUT o in quelle di assegnazione delle variabili (campo = valore), dato che esse riallocano il puntatore del campo nello spazio consueto della stringa, in modo che il campo diventi una normale variabile. Tuttavia, è possibile far riferimento senza alcun problema alle variabili definite nelle istruzioni FIELD nel caso di un'espressione logica o aritmetica.

Commenti

L'istruzione FIELD può definire soltanto campi che sono stringhe a lunghezza variabile; per utilizzare le variabili numeriche è necessario effettuare altre operazioni per convertire i numeri nei corrispondenti equivalenti di stringa e quindi di nuovo in numeri. Le funzioni MKD$, MKDMBF$, MKI$, MKL$, MKS$ e MKSMBF$ convertono i numeri in stringhe, mentre CVD, CVDMBF, CVI, CVL, CVS e CVSMBF convertono nuovamente queste stringhe in numeri. L'utilizzo dell'istruzione TYPE...END TYPE per definire un record consente di eliminare la necessità di eseguire questa procedura complicata.

Errori

La somma delle dimensioni dei campi definiti in un'istruzione FIELD non deve superare la lunghezza del record specificata nella clausola AS, quando il file viene aperto, altrimenti QBasic visualizza il messaggio di errore: "Overflow dell'istruzione FIELD".

Esempio

Il programma di seguito riportato utilizza due file ad accesso casuale che comprendono i record definiti dalle istruzioni FIELD e copia i dati da un file meno recente in un nuovo file che utilizza una struttura record diversa.

```
OPEN "PAZIENTI.OLD" FOR RANDOM AS #1 LEN = 128
FIELD #1, 25 AS vNomePaziente$, 25 AS vNomeAssicurato$

OPEN "PAZIENTI.NEW" FOR RANDOM AS #2 LEN = 256
FIELD #2, 32 AS nNomePaziente$, 32 AS nNomeAssicurato$

IF LOF(1) > 0 THEN
   FOR numRecord% = 1 TO LOF(1) \ 128          'divisione di tipo intero
       GET #1, numRecord%
       LSET nNomePaziente$ = vNomePaziente$     'assegna informazioni dal
       LSET nNomeAssicurato$ = vNomeAssicurato$ 'vecchio file al nuovo
       PUT #2, numRecord%
   NEXT

   PRINT "Conversione completata."
ELSE
   PRINT "Non ci sono record in PAZIENTI.OLD da convertire."
END IF

CLOSE
```

FILEATTR

Si consulti anche

OPEN

Scopo

La funzione FILEATTR restituisce la modalità del file o il file handle del DOS di un file aperto; può essere utilizzata, ad esempio, per determinare se un file è a sola lettura, in modo tale che il programma in questo caso non cerchi di scrivervi dati.

Sintassi

```
FILEATTR(numerofile,attributo)
```

numerofile è il numero di un file aperto.

attributo può essere 1 o 2. Se attributo è 1, FILEATTR restituisce le informazioni riguardanti la modalità in base alla quale è stato aperto un file; se attributo è 2, FILEATTR restituisce il file handle del DOS.

Utilizzo

```
OPEN "INFO.TXT" FOR INPUT AS #1
PRINT "La modalità di INFO.TXT è"; FILEATTR(1, 1)
```

Visualizza il numero 1, che indica che il file è stato aperto in modalità input.

Descrizione

Il numero del file utilizzato per aprirlo in QBasic non è direttamente collegato al numero di accesso del DOS, il file handle. Tuttavia, alcune subroutine scritte in linguaggio assembler richiedono che venga fornito questo valore. FILEATTR fornisce inoltre informazioni per i programmi che necessitano di conoscere la modalità del file per determinare il tipo di I/O da eseguire.

Per ottenere la modalità di apertura del file, è sufficiente utilizzare il numero 1 come secondo parametro di FILEATTR. Ad esempio:

```
ModFile% = FILEATTR(4, 1)
```

restituisce il valore che indica la modalità del file aperto come numero 4. Il valore restituito è uno di quelli elencati nella Tabella 17.4. Ad esempio, se a ModFile% è stato assegnato il valore 32 nell'istruzione precedente, ciò significa che il file è stato originariamente aperto come binario.

Per ottenere il file handle del DOS, è sufficiente utilizzare il numero 2 come secondo parametro di FILEATTR. Ad esempio:

```
fileHandle% = FILEATTR(4, 2)
```

restituisce il file handle del DOS relativo al file aperto come numero 4.

Errori

Al contrario della maggiore parte delle istruzioni di I/O di QBasic, FILEATTR non accetta il segno di numero (#) con il numero del file. In questo caso, QBasic visualizza il messaggio di errore: "Era atteso: espressione".

Se la funzione FILEATTR viene utilizzata con un numero di file che non è stato specificato da un'istruzione OPEN, QBasic visualizza il messaggio di errore: "Nome o numero di file errato".

Tabella 17.4 I valori restituiti da FILEATTR

Valore restituito	Modalità
1	Input
2	Output
4	Random
8	Append
32	Binary

Esempio

Il programma di seguito riportato utilizza la funzione FILEATTR per determinare il file handle del DOS e la modalità di apertura.

```
CLS
OPEN "PIPPO.DAT" FOR APPEND AS #7

modoFile% = FILEATTR(7, 1)
numFile% = FILEATTR(7, 2)

PRINT "PIPPO.DAT ha un numero file DOS "; numFile%; " ed è stato aperto come
un ";

SELECT CASE modoFile%
    CASE 1
         PRINT "file sequenziale di sola lettura. ";
    CASE 2
         PRINT "file sequenziale di sola scrittura. ";
    CASE 4
         PRINT "file ad accesso casuale di lettura o scrittura. ";
    CASE 8
         PRINT "file sequenziale e il puntatore di file è stato spostato alla
         fine del file. ";
    CASE 32
         PRINT "file binario di lettura o scrittura. ";
END SELECT

CLOSE
```

FILES

Si consulti anche

CHDIR

Scopo

L'istruzione FILES visualizza i nomi dei file che corrispondono alle specifiche fornite; può essere utilizzata per visualizzare tutti o solo alcuni dei file presenti nella directory attuale o in quella specificata.

Sintassi

```
FILES [specfile]
```

specfile è un'espressione stringa che indica i file da visualizzare. Questo parametro può anche contenere il nome dell'unità, il percorso e il nome del file, che può comprendere anche i caratteri jolly del DOS ? e *.

Utilizzo

```
FILES
```

Visualizza i nomi di tutti i file presenti nella directory in uso (nell'unità attiva).

```
FILES "C:\BIN\*.*"
```

Visualizza i nomi di tutti i file presenti nella directory BIN, nell'unità disco C.

Descrizione

L'istruzione FILES è analoga al comando DIR/W del DOS, dato che consente di visualizzare i nomi dei file nella directory attiva o i file specificati suddivisi in colonne. Se tutti i programmi d'esempio illustrati in questo capitolo vengono inseriti nella directory CAP17 sull'unità disco C, l'istruzione FILES "C:\CAP17*.*" visualizza il seguente elenco:

```
C:\CAP17
     .   <DIR>              ..  <DIR> CLOSE    .BAS          CVD      .BAS
EOF      .BAS        FIELD    .BAS    FILEATTR.BAS           FILES    .BAS
FREELINE.BAS         GET      .BAS    INPUT#   .BAS          INPUT#   .BAS
LINEINPT.BAS         LOC      .BAS    LOCK     .BAS          LOF      .BAS
LTRIM$   .BAS        NAME     .BAS    OPEN     .BAS          PRINT#   .BAS
PRT#USNG.BAS         PUT      .BAS    RESET    .BAS          RSET     .BAS
RTRIM$   .BAS        SEEK(FN).BAS     SEEK(ST).BAS           UNLOCK   .BAS
WIDTH    .BAS        WRITE#   .BAS
 19511296 byte disponibili
```

Il parametro di specifica dei file è un'espressione stringa facoltativa che indica i file da visualizzare; può anche contenere il nome dell'unità, il percorso, il nome dei file, che può comprendere anche i caratteri globali del DOS ? e *. Nel DOS, i nomi dei file possono essere composti da un massimo di otto caratteri, da un punto e da tre caratteri per l'estensione. Il carattere jolly ? corrisponde a uno o più caratteri che si trovano nella stessa posizione nel nome del file.

Se viene tralasciato il nome dell'unità disco, l'istruzione FILES utilizza quello attivo; se viene omesso il nome del percorso, FILES utilizza la directory in uso; se non viene inserita alcuna specifica, FILES visualizza tutti i file presenti nella directory attiva dell'unità in uso.

Al contrario del comando DIR/W del DOS, l'istruzione FILES visualizza un'intestazione composta da una riga che contiene la directory attiva, anche se nell'istruzione viene indicato un percorso diverso. Dopo aver visualizzato tutti i nomi di file che corrispondono alle specifiche immesse, FILES visualizza una riga che contiene il numero di byte liberi rimasti.

Commenti

I caratteri jolly possono essere utilizzati soltanto nel nome del file o nelle specifiche; QBasic non li riconosce nel nome dell'unità o nel percorso.

Se un listato contiene più di una singola schermata di nomi di file, questa istruzione fa scorrere i primi nomi nella parte superiore dello schermo.

Errori

Se la specifica di file non corrisponde a nessun file QBasic visualizza il messaggio di errore: "File non trovato". Se viene indicata la lettera di un'unità che non contiene un disco, QBasic visualizza il messaggio di errore: "Disco non pronto". Se viene specificata un'unità che non esiste, QBasic visualizza il messaggio di errore: "Periferica non disponibile".

Esempio

Il programma di seguito riportato utilizza l'istruzione FILES per visualizzare un elenco dei file della directory e dell'unità indicati dall'utente. Inoltre, esegue un'intercettazione degli errori, in modo che se un utente immette delle specifiche errate il programma non termina.

```
ON ERROR GOTO ControllaErr

DO
   PRINT "Immettere un indicatore di drive, un percorso e un nome file."
   PRINT "(il nome file può includere caratteri sostitutivi)"
   PRINT
   LINE INPUT nomeFile$

   IF nomeFile$ = "" THEN EXIT DO

   PRINT
   FILES nomeFile$
   PRINT
LOOP UNTIL nomeFile$ = ""
END

'gestione degli errori di input dell'utente
ControllaErr:
   numErr% = ERR
   SELECT CASE numErr%
       CASE 53
            PRINT "Il file non esiste."
            RESUME NEXT
       CASE 64
            PRINT "Il nome file ha un formato errato."
            RESUME NEXT
       CASE 68
            PRINT "Il dispositivo non è disponibile."
            RESUME NEXT
       CASE 71
            PRINT "Nessun disco inserito nel drive."
            RESUME NEXT
       CASE ELSE
            ON ERROR GOTO 0            'disabilita la gestione degli errori
                                       'per tutti gli altri errori
   END SELECT
```

FREEFILE

Si consulti anche

OPEN

Scopo

La funzione FREEFILE restituisce il più basso numero di file disponibile, non associato con un file aperto; può essere utilizzata quando è necessario aprire un file, ma non si conoscono i numeri dei file disponibili.

Sintassi

```
FREEFILE
```

Utilizzo

```
NumFile% = FREEFILE
OPEN "CONFIG.DAT" FOR RANDOM AS #NumFile%
```

Apre un file, assegnandogli il primo numero di file disponibile.

Descrizione

Per accedere ai file di QBasic è necessario utilizzare un numero nell'intervallo da 1 a 255, ma non è possibile indicarne uno che è già stato assegnato a un file in un'istruzione OPEN. FREEFILE risolve questo problema, restituendo il numero del file disponibile più basso (da 1 a 255). A questo punto è sufficiente assegnare il valore restituito da FREEFILE a una variabile numerica, da utilizzare nelle successive istruzioni OPEN, CLOSE e altre di I/O.

Avvertenze

QBasic non consente di aprire più di 15 file alla volta; se si cerca di aprire il sedicesimo file, QBasic visualizza il messaggio di errore: "Troppi file".

Esempio

Il programma di seguito riportato utilizza la funzione FREEFILE per assegnare i numeri dei file ai cinque file di dati.

```
OPEN "BUFF1.DAT" FOR OUTPUT AS #1

FOR i% = 1 TO 5
    numFile% = FREEFILE
    nomeFile$ = "BUFF" + MID$(STR$(numFile%), 2) + ".DAT"
    OPEN nomeFile$ FOR OUTPUT AS #numFile%
    PRINT nomeFile$
NEXT i%
CLOSE
```

GET (I/O su file)

Si consulti anche

FIELD, INPUT #, LINE INPUT #, OPEN, PUT (I/O su file), TYPE...END TYPE

Scopo

L'istruzione GET legge i dati da un file binario o ad accesso casuale e li memorizza nei campi definiti dall'istruzione FIELD o in una variabile del record definita da un'istruzione TYPE...END TYPE.

Sintassi

```
GET[#]numerofile[,[numerorecord][,var]]
```

numerofile è il numero di un file aperto.

numerorecord è il numero del record (nei file ad accesso casuale) o la posizione del byte (nei file in formato binario), in cui inizia la lettura o la scrittura; è compreso nell'intervallo da 1 a 2147483647.

var è la variabile che riceve i dati da un file.

Utilizzo

```
GET #1, 1
```

Legge il primo record in un file ad accesso casuale che contiene record definiti da un'istruzione FIELD.

```
GET #1, , info
```

Legge il record successivo (se presente) in un file ad accesso casuale, assegnandolo alla variabile record info.

Descrizione

Nel caso dei file ad accesso casuale o in formato binario, è necessario utilizzare GET per leggere i dati.

Quando questa istruzione viene utilizzata con un file ad accesso casuale, è possibile specificare il numero del record da leggere. Ogni record comprende un numero di byte uguale alla lunghezza del record indicata nella clausola LEN dell'istruzione OPEN relativa a quel file. Il primo numero di record in un file è 1; se non viene specificato alcun numero, GET legge il record successivo a quello utilizzato nell'ultima operazione GET o PUT, oppure quello indicato nell'operazione SEEK più recente.

Quando GET viene utilizzata con un file binario, è possibile specificare il numero del byte in cui inizia la lettura. Il primo byte in un file è il numero 1; se non viene specificato alcun numero, GET legge il byte successivo a quelli utilizzati nell'ultima operazione GET o PUT, oppure quello indicato nell'operazione SEEK più recente.

Nel caso dei file ad accesso casuale che contengono record definiti da un'istruzione TYPE...END TYPE, è possibile indicare qualsiasi variabile le cui dimensioni in byte non siano superiori alla lunghezza del record indicata quando il file è stato aperto; l'istruzione GET legge i dati in questa variabile. Ad esempio, se la variabile è un intero lungo, l'istruzione GET legge 4 byte dal file nella variabile. Molto spesso viene specificata una variabile record la cui struttura corrisponde a quella dei record del file.

Se viene utilizzata una variabile in un'istruzione GET, e i record nel file erano stati definiti da un'istruzione FIELD, QBasic visualizza il messaggio di errore: "Istruzione FIELD attiva". Se non viene specificata una variabile, GET presuppone che i record contenuti nel file siano stati definiti da un'istruzione FIELD. Dopo aver richiamato GET, è possibile accedere ai valori presenti in altre variabili utilizzando l'istruzione INPUT # o LINE INPUT #.

Quando GET viene utilizzata con un file in formato binario, è possibile specificare una variabile di qualsiasi tipo e lunghezza. Fino a quando i byte nel file sono un numero sufficiente, GET legge quelli necessari a riempire la variabile.

Quando in un'istruzione GET viene indicata una variabile stringa a lunghezza variabile, GET legge soltanto il numero di caratteri presenti in essa; ad esempio, se si utilizza una variabile stringa che ha cinque caratteri, GET li legge nella variabile, sovrascrivendo quelli originali.

Dato che la dimensione massima possibile di una stringa o di una variabile record è 32767 byte, questo valore è anche il valore massimo di ogni record in un file e della variabile che QBasic accetta per l'istruzione **GET** da un file ad accesso casuale o in formato binario.

Suggerimenti

Se viene specificato un record o un numero di byte oltre la fine del file, QBasic non restituisce un messaggio di errore. Se viene indicato un record o un byte che si trova oltre la fine di un file binario o ad accesso casuale, **GET** inserisce nella variabile indicata dagli spazi vuoti, oppure la imposta a 0. Per verificare esplicitamente se il programma ha raggiunto la fine del file, è sufficiente utilizzare **EOF** o **LOF**

Avvertenze

Anche se le istruzioni **GET** e **PUT** consentono un accesso a lunghezza fissa da un file aperto come dispositivo di comunicazione, quando si verifica un malfunzionamento **GET** attende i caratteri per un tempo indeterminato; perciò è meglio utilizzare l'istruzione **INPUT #** o la funzione **INPUT$** per leggere i dati da un dispositivo di comunicazione.

Esempio

Il programma di seguito riportato crea una variabile record che memorizza le scelte dei colori utilizzati nel programma. Questo esempio utilizza **PUT** per memorizzare e **GET** per caricare la variabile da un file di configurazione.

```
CLS
TYPE TipoConfig
     normale AS INTEGER          'colore dell'area interna del riquadro
     altaInt AS INTEGER          'colore della lettera in alta intensità
     invAltaInt AS INTEGER       'colore dell'inverso della lettera in alta
                                 'intensità
     inverso AS INTEGER          'colore della barra in alta intensità
     aiuto AS INTEGER            'colore della riga di aiuto
     cursoreNorm AS INTEGER      'dimensione del cursore in modo sovrascrittura
     cursoreIns AS INTEGER       'dimensione del cursore in modo Inserimento
END TYPE
DIM config AS TipoConfig         'definisce la variabile record config

OPEN "CONFIG.DAT" FOR BINARY AS #1        'apre il file
     IF LOF(1) = 0 THEN         'imposta i colori di default se il file è vuoto
        config.normale = 7
        config.altaInt = 15
        config.invAltaInt = 126
        config.inverso = 112
        config.aiuto = 1
        config.cursoreNorm = 7
        config.cursoreIns = 15
        PUT #1, 1, config   'scrive i valori di default nel file
     ELSE
        GET #1, 1, config 'legge le informazioni sulla configurazione attuale
     END IF
CLOSE #1

'visualizza informazioni sulla configurazione corrente
PRINT "Colore normale = "; config.normale
PRINT "Colore alta intensità = "; config.altaInt
```

```
PRINT "Inverso colore alta intensità = "; config.invAltaInt
PRINT "Inverso colore = "; config.inverso
PRINT "Colore aiuto = "; config.aiuto
PRINT "Dimensione del cursore in modalità sovrascrittura = ";
config.cursoreNorm
PRINT "Dimensione del cursore in modalità inserimento = "; config.cursoreIns

END
```

INPUT

Si consulti anche

INKEY$, INPUT, INPUT$ (I/O su file), LINE INPUT, LINE INPUT #

Scopo

L'istruzione INPUT # legge i valori da un file sequenziale o da un dispositivo e li assegna alle variabili; può essere utilizzata per leggere i dati formattati da un file ASCII o i dati ASCII da un dispositivo.

Sintassi

```
INPUT #numerofile, var[var...]
```

numerofile è il numero del file utilizzato per aprire il file sequenziale.
var è la variabile numerica o stringa che riceve i dati dal file.

Utilizzo

```
INPUT #1, titolo$
```

Legge un valore dal file sequenziale associato al numero 1 e lo assegna alla variabile stringa titolo$.

```
INPUT #10, x%, y%, z%
```

Legge i valori dal file sequenziale associato al numero 10 e li assegna alle variabili di tipo intero x%, y% e z%.

Descrizione

L'istruzione INPUT # è utile per leggere le variabili numeriche dai file sequenziali; quando assegna i dati alle variabili specificate, ignora gli spazi iniziali, i codici di ritorno a capo e di avanzamento riga. INPUT # interrompe la lettura di un numero quando individua uno spazio, un ritorno a capo, un avanzamento riga o la fine del file; interrompe la lettura di una stringa quando incontra punti di domanda, ritorni a capo, avanzamenti riga o la fine del file.

È possibile specificare una variabile semplice, un elemento di array, o un elemento di una variabile record. In una istruzione è possibile indicare diverse variabili, separandole con una virgola.

L'istruzione INPUT # legge i dati nel formato ASCII, come se fossero stati immessi da tastiera, in risposta all'istruzione INPUT.

Nella Figura 17.1 è illustrato il contenuto di un file sequenziale.

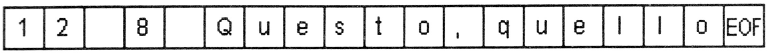

Figura 17.1 Un file sequenziale

Dopo aver eseguito le istruzioni INPUT # di seguito riportate, in cui il numero 1 specifica il file riportato nella Figura 17.1, x% e y% contengono i valori 12 e 8, la variabile a$ contiene la stringa "Questo" e b$ "quello".

```
INPUT #1, x%, y%
INPUT #1, a$
INPUT #1, b$
```

Nella Figura 17.2 è riportata la posizione del puntatore del file, dopo l'esecuzione delle precedenti istruzioni.

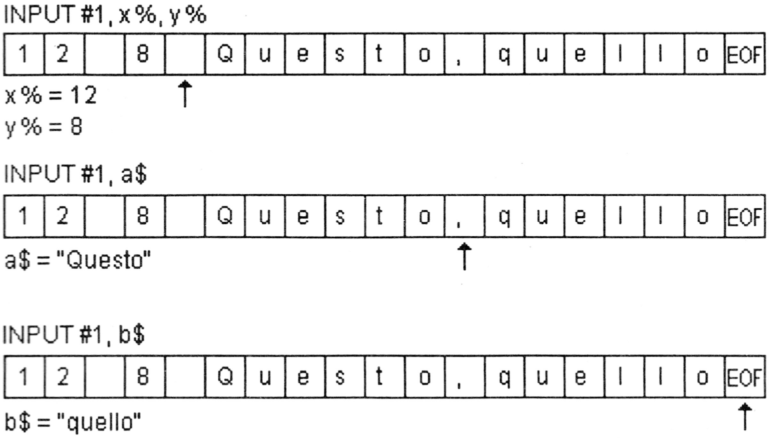

Figura 17.2 L'utilizzo di INPUT # per leggere un file sequenziale

Commenti

Se INPUT # legge un numero in virgola mobile (contenente una parte decimale) da un file sequenziale e lo assegna a una variabile di tipo intero o intero lungo, tale numero viene arrotondato per adattarsi al tipo all'intero.

Utilizzando INPUT # per leggere un carattere alfabetico e assegnandolo a una variabile numerica non si provoca alcun errore, tuttavia QBasic imposta la variabile numerica a 0.

Avvertenze

Dato che INPUT # utilizza le virgole per separare le voci di dati, una stringa non può contenere virgole, a meno che non sia tutta racchiusa tra doppi apici. Ad esempio,

Arrivederci, a domani viene letta come due stringhe: *Arrivederci* e *a domani*, mentre *"Arrivederci, a domani"* viene letta come una stringa.

Esempio

Il programma di seguito riportato scrive in un file una riga di testo e i valori di 2 elevato agli esponenti da 1 a 25, utilizzando quindi INPUT # per leggere il contenuto del file.

```
CLS
OPEN "2EXP.DAT" FOR OUTPUT AS #1

WRITE #1, "2EXP.DAT contiene il valore di 2 elevato a potenza da 1 a 25."
FOR i% = 1 TO 25
    risultato& = 1
    FOR j% = 1 TO i%
        risultato& = risultato& * 2
    NEXT j%
    WRITE #1, risultato&
NEXT i%
CLOSE #1

OPEN "2EXP.DAT" FOR INPUT AS #1

INPUT #1, descrizione$
PRINT descrizione$

PRINT "Premi un tasto qualsiasi per continuare."
DO: LOOP UNTIL INKEY$ <> ""

FOR i% = 1 TO 25
    INPUT #1, valore&
    PRINT valore&
NEXT i%
CLOSE #1
```

INPUT$ (I/O su file)

Si consulti anche

INPUT, LINE INPUT

Scopo

La funzione INPUT$ legge un numero specificato di byte da tastiera, da un file o da un dispositivo; può essere utilizzata per controllare la quantità dell'input.

Sintassi

```
INPUT$(numero[,[#]numerofile])
```

numero è il numero di caratteri da leggere (la stringa più ampia che INPUT$ è in grado di leggere è di 32767 byte).

numerofile è il numero che può essere associato al file o al dispositivo da cui vengono letti i caratteri.

Utilizzo

```
PRINT "Premere un tasto per continuare: "; INPUT$(1);
```

Richiede all'utente di premere un tasto e quindi utilizza INPUT$ per leggerlo.

Descrizione

L'istruzione INPUT$ è utile quando un programma deve leggere un numero specificato di caratteri, invece di elaborare stringhe (come fanno INPUT # e LINE INPUT #). È necessario tuttavia tenere presente che INPUT$ non rappresenta la scelta migliore, se non si conosce il numero di caratteri da leggere.

Se viene specificato un numero di file, INPUT$ legge i caratteri dal file o dal dispositivo associati; se questo numero viene omesso, l'istruzione legge i caratteri da tastiera. Se il file è stato aperto come file ad accesso casuale, il numero di caratteri che INPUT$ legge non può essere superiore a quello presente in ogni record specificato nell'istruzione OPEN per quel file (nella clausola LEN). Se il file è stato aperto in formato binario o sequenziale, INPUT$ non può leggere più di 32767 caratteri.

INPUT$ non visualizza qualsiasi carattere che legge da tastiera, da file o da un dispositivo, operazione che ovviamente può essere eseguita utilizzando PRINT. Si noti che INPUT$ legge tutti i caratteri, tranne la combinazione CTRL+INTERR, che interrompe il programma.

Errori

Se viene specificato un numero di file relativo a un file che non è aperto, QBasic visualizza il messaggio di errore: "Nome o numero di file errato". Si noti che il numero di caratteri deve essere disponibile dal file specificato; se quest'ultimo non contiene un numero sufficiente di caratteri, QBasic visualizza il messaggio di errore: "Input oltre la fine del file". Se il numero di caratteri indicati per un file ad accesso casuale è superiore alla lunghezza del record, QBasic visualizza il messaggio di errore: "Overflow dell'istruzione FIELD". Se il numero di caratteri specificato per un file in formato binario o sequenziale è maggiore di 32767, QBasic visualizza il messaggio di errore: "Overflow".

Suggerimenti

La funzione INPUT$ è il metodo preferito per l'input e l'output dei dispositivi di comunicazione, dato che consente di indicare il numero di caratteri da caricare. Quando viene utilizzata con la funzione LOC, INPUT$ è in grado di gestire anche le comunicazioni in entrata.

Avvertenze

Se viene omesso il numero del file, il numero di caratteri specificati deve essere immesso da tastiera. Il tasto INVIO non indica la fine del file, come nelle istruzioni INPUT e LINE INPUT.

Esempio

Il programma di seguito riportato utilizza INPUT$ per caricare da un file una password composta da 8 caratteri, accettando quindi una password di 8 caratteri da tastiera. Se le due password non corrispondono, il programma visualizza il messaggio: "Accesso negato".

```
OPEN "PASSWORD.DAT" FOR INPUT AS #1

password$ = INPUT$(8, #1)              'legge una password dal file

PRINT "Inserire una password:";
pswUtente$ = INPUT$(8)                 'legge una password dalla tastiera

IF password$ = pswUtente$ THEN
   PRINT "Accesso accordato."
ELSE
   PRINT "Accesso negato."
   STOP
END IF
CLOSE
```

LINE INPUT

Si consulti anche

INPUT, INPUT #, INPUT$ (I/O su file), LINE INPUT

Scopo

L'istruzione LINE INPUT # legge i dati da un file sequenziale o da un dispositivo e li assegna a una variabile stringa. Al contrario dell'istruzione LINE #, che legge le stringhe separate da delimitatori, LINE INPUT # legge riga per riga il contenuto di un file sequenziale. Ad esempio, questa istruzione può essere utilizzata per leggere le istruzioni da un file ASCII.

Sintassi

```
LINE INPUT #numerofile, var
```

numerofile è il numero utilizzato per aprire il file sequenziale.
var è la variabile stringa che riceve i dati letti dal file.

Utilizzo

```
LINE INPUT #1, titolo$
```

Legge una riga dal file associato al numero 1 e la assegna alla variabile stringa titolo$.

Descrizione

L'istruzione INPUT # è utile per leggere una riga di testo da un file sequenziale o da una dispositivi. Questa istruzione legge tutti i caratteri presenti in una riga, ma non è in grado di leggere più di 32767 caratteri, ovvero il numero massimo di caratteri che può essere assegnato a una variabile stringa. INPUT # ignora gli spazi iniziali, ma si interrompe quando individua una virgola o la coppia di codici ritorno a capo/

avanzamento riga. L'istruzione LINE INPUT # non ignora gli spazi iniziali e legge tutti i caratteri, comprese le virgole, fino ai due codici di ritorno a capo/avanzamento riga che indicano la fine della riga. L'istruzione INPUT # viene spesso utilizzata per leggere un file di testo da un programma di elaborazione testi o da un database. Se utilizzando il comando TYPE del DOS è possibile leggere i caratteri contenuti in un file, l'istruzione LINE INPUT # può essere sfruttata per leggere il file.

Nella Figura 17.3 è illustrato il contenuto di un file sequenziale; le iniziali CR e LF rappresentano i caratteri di ritorno a capo e avanzamento riga.

Dopo l'esecuzione delle seguenti istruzioni, in cui il numero 1 specifica il file riportato nella Figura 17.3, la variabile x$ contiene la stringa "12 8" e la variabile a$ contiene la stringa "Questo, quello".

```
LINE INPUT #1, x$
LINE INPUT #1, a$
```

Nella Figura 17.4 è riportata la posizione del puntatore del file, dopo l'esecuzione delle precedenti istruzioni.

Commenti

L'unico modo per delimitare un testo in un file che deve essere letto da LINE INPUT # consiste nell'inserire i dati su righe separate, infatti questa istruzione legge le virgole poste a metà delle stringhe come caratteri, non come delimitatori.

Suggerimenti

Dato che le variabili stringa possono contenere qualsiasi carattere, comprese le cifre, quando viene letto un numero da un file non viene visualizzato alcun messaggio di errore, ma LINE INPUT # si limita ad assegnare le cifre alla variabile stringa. VAL può essere utilizzato per convertire le cifre in variabili numeriche.

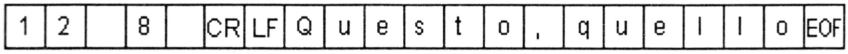

Figura 17.3 Un file sequenziale

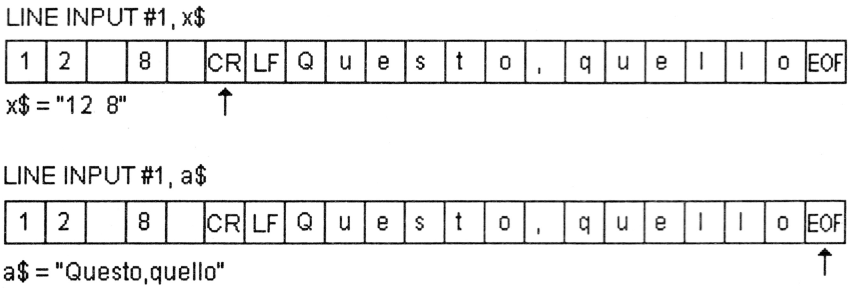

Figura 17.4 L'utilizzo di LINE INPUT # per leggere un file sequenziale

Esempio

Il programma di seguito riportato illustra alcune diversità tra l'utilizzo di INPUT # e di LINE INPUT # con il testo. Si noti che quest'ultima istruzione restituisce il file riga per riga, proprio come quando viene visualizzato con il comando TYPE del DOS.

```
OPEN "TESTO.DAT" FOR INPUT AS #1

PRINT "Esempio di INPUT # :"
PRINT
numero = 0
DO
  numero = numero + 1
  INPUT #1, testo$
  PRINT numero; testo$
LOOP UNTIL EOF(1)

PRINT
PRINT "Esempio di LINE INPUT # :"
PRINT
numero = 0
SEEK #1, 1                    'sposta il puntatore all'inizio del file
DO
  numero = numero + 1
  LINE INPUT #1, testo$
  PRINT numero; testo$
LOOP UNTIL EOF(1)

CLOSE
```

LOC

Si consulti anche

GET (I/O su file), OPEN, PUT (I/O su file)

Scopo

QBasic mantiene per ogni file aperto un puntatore che indica il punto in cui viene effettuata la successiva operazione di lettura o di scrittura. Dato che la funzione LOC restituisce la posizione attuale di questo puntatore in un file aperto, può essere utilizzata per determinare se la posizione all'interno del file è corretta, prima di eseguire un'istruzione PUT o GET.

Sintassi

```
LOC(numerofile)
```

numerofile è il numero del file con il quale esso è stato aperto.

Utilizzo

```
PRINT "La posizione attuale nel file 1 è"; LOC(1)
```

Visualizza a quale posizione nel file accede la successiva istruzione di I/O.

Descrizione

Le istruzioni GET e PUT consentono di eseguire le operazioni di lettura e di scrittura di un file. La funzione LOC può essere utilizzata per verificare la posizione attuale, in modo che non venga letto il record sbagliato o sovrascritto un altro record.

I valori restituiti da LOC dipendono dalla modalità in base alla quale è stato aperto il file. Se questo è stato aperto nel formato ad accesso casuale, LOC restituisce il numero dell'ultimo record che è stato letto o scritto.

Se il file è stato aperto nel formato sequenziale, LOC restituisce la posizione attuale in numero di byte diviso per 128 e arrotondato a un intero. Dato che il buffer di QBasic ha una lunghezza di 128 byte, questa operazione restituisce la posizione attuale nel buffer.

Se il file è stato aperto nel formato binario, LOC restituisce il numero dell'ultimo byte scritto o letto. Ad esempio, se il programma legge da un file binario una variabile stringa, denominata titolo$, lunga 100 byte, LOC restituisce 100; la successiva posizione del byte è 101.

Se il file è un dispositivo di comunicazione (COM1: o COM2:) aperto in modalità binaria, LOC restituisce il numero dei caratteri restanti in quel buffer di input. Se il file è un dispositivo di comunicazione aperto in qualsiasi altra modalità, LOC restituisce il numero dei caratteri restanti in quel buffer di input fino a, ma non compreso, il carattere di fine file Ctrl+Z (questa informazione è importante, dato che, se si cerca di immettere i caratteri oltre il contrassegno di fine file, QBasic visualizza il messaggio di errore: "Input oltre la fine del file").

Commenti

Prima di scrivere o leggere un file, la funzione LOC restituisce il valore 0, dato che non è stata effettuata nessuna operazione di input o di output. Il valore massimo che può essere restituito da LOC è 2147483647.

Avvertenze

La funzione LOC restituisce 0, quando viene utilizzata con i dispositivi SCRN:, KYBD:, CONS: o LPTn. Questa funzione restituisce valori significativi soltanto quando viene sfruttata con un file o con il dispositivo COMn:.

Esempio

Il programma di seguito riportato utilizza la funzione LOC per determinare il numero di byte utilizzati in ogni operazione di output in un file binario.

```
CLS

OPEN "EX-LOCK.DAT" FOR BINARY AS #1

FOR dimensioni% = 1 TO 23
    currPos& = LOC(1) + 1       'trova la posizione corrente nel file
    nuovaPos& = currPos& + dimensioni% 'trova la posizione successiva nel file

    LOCK #1, currPos& TO nuovaPos&          'impedisce l'accesso a una
                                            'porzione del file
    varStringa$ = STRING$(dimensioni%, "*")
```

```
    PUT #1, , varStringa$             'scrive la variabile nel file
    UNLOCK #1, currPos& TO nuovaPos& 'rimuove le limitazioni di accesso alla
                                      'porzione del file

    PRINT "Aggiunge asterischi al file dalla posizione ";
    PRINT USING "#### alla posizione ####."; currPos&; nuovaPos&
NEXT

CLOSE
```

LOCK

Si consulti anche

OPEN, UNLOCK

Scopo

L'istruzione LOCK limita o impedisce l'accesso a un file condiviso in una rete. In un sistema multitasking, multiutente o di rete, a un file condiviso possono accedere diversi utenti o elaborazioni: questa istruzione può essere utilizzata per impedire l'accesso a un record, dopo che il proprio programma lo ha modificato.

Sintassi

```
LOCK [#]numerofile [,{record | [inizio] TO fine}]
```

numerofile è il numero di un file aperto.
record è il numero del record o del byte da bloccare.
inizio e fine sono interi che indicano l'intervallo dei record o dei byte da bloccare.

Utilizzo

```
LOCK #1
```

Blocca tutto il file aperto come numero 1. Fino a quando il programma non sblocca il file utilizzando l'istruzione UNLOCK, non è possibile accedervi.

```
OPEN "VENDITE.DAT" FOR RANDOM AS #4
LOCK #4, 10
```

Blocca il record 10 del file ad accesso casuale aperto come numero 4.

Descrizione

In un ambiente di rete in cui diversi utenti od operazioni condividono gli stessi file, è necessario impedire che elaborazioni differenti aggiornino contemporaneamente lo stesso record, in caso contrario rimane valida soltanto un'operazione di aggiornamento, mentre le modifiche apportate da altre elaborazioni vengono sovrascritte. L'istruzione LOCK impedisce che ciò avvenga, permettendo soltanto a un'operazione di avere i diritti esclusivi su un file o un record e negando l'accesso ad altre operazioni. Un'istruzione LOCK deve sempre essere seguita da una corrispondente istruzione UNLOCK.

In un file binario è necessario specificare i byte da bloccare; il primo byte è quello numero 1. In un file ad accesso casuale, è sufficiente specificare i record; il primo record è quello numero 1. Se viene specificato soltanto un record o un numero di byte, soltanto quel record o quel byte vengono bloccati per poter essere utilizzati esclusivamente dal programma. Se viene inserita la parola chiave TO, vengono invece bloccati i record o i byte compresi nell'intervallo specificato. Se viene omesso il numero relativo all'inizio dell'intervallo, LOCK utilizza il record o il byte 1 come posizione iniziale di default; se non viene specificato alcun record o byte, l'istruzione LOCK blocca tutto il file, in modo che possa essere utilizzato esclusivamente dal proprio programma.

Nei file ad accesso casuale o binari è possibile bloccare singoli record o byte, mentre nei file sequenziali può essere bloccato soltanto tutto il file.

LOCK funziona al momento dell'esecuzione soltanto quando è attivato SHARE.EXE, il programma fornito con il DOS che supporta la condivisione e il bloccaggio dei file per le reti Microsoft.

Errori

È possibile accedere alle parti bloccate dei file soltanto dal programma che esegue l'istruzione LOCK. Se un altro programma QBasic cerca di eseguire questa operazione, QBasic visualizza il messaggio di errore: "Permesso negato".

Avvertenze

Microsoft supporta QBasic soltanto sulle reti Microsoft LAN Manager, IBM-PC Network, MS-Network e sistemi compatibili. Sebbene siano stati ottenuti dei risultati anche con altre reti, quali Novell e 3Com, possono in ogni caso sorgere dei problemi.

Se dopo che un file viene bloccato si verifica un calo di tensione o si esegue il reset del computer, prima che venga eseguita la corrispondente istruzione UNLOCK, si potrebbero perdere i dati presenti nel file bloccato. Per ridurre al minimo la possibilità di perdere dati è necessario utilizzare LOCK per bloccare i record solo fino a quando è strettamente necessario.

Esempio

Il programma di seguito riportato utilizza le istruzioni LOC e LOCK per proteggere alcune parti di un file binario, mentre il programma le aggiorna.

```
CLS

OPEN "EX-LOCK.DAT" FOR BINARY AS #1

FOR dimensioni% = 1 TO 23
  currPos& = LOC(1) + 1  'trova la posizione corrente nel file
  nuovaPos& = currPos& + dimensioni% 'trova la posizione successiva nel file

  LOCK #1, currPos& TO nuovaPos& 'impedisce l'accesso ad una porzione del file
  varStringa$ = STRING$(dimensioni%, "*")
  PUT #1, , varStringa$             'scrive la variabile nel file
  UNLOCK #1, currPos& TO nuovaPos&        'rimuove le limitazioni di accesso
                                          'alla porzione del file

  PRINT "Aggiunge asterischi al file dalla posizione ";
  PRINT USING "#### alla posizione ####."; currPos&; nuovaPos&
NEXT

CLOSE
```

LOF

Si consulti anche

LOC, OPEN

Scopo

La funzione LOF restituisce il numero di byte in un file aperto; può essere utilizzata quando è necessario conoscere la lunghezza di un file, prima di iniziare ad elaborarlo.

Sintassi

```
LOF(numerofile)
```

numerofile è il numero del file aperto.

Utilizzo

```
PRINT "Il file è lungo"; LOF(1); "byte."
```

Visualizza un messaggio che fornisce la lunghezza del file aperto come numero 1.

Descrizione

LOF deve essere utilizzata per determinare le dimensioni di un file aperto; restituisce le dimensioni, in byte, di un file aperto in qualsiasi modalità. Il valore massimo che può essere restituito è 2147483647.

Se il file aperto è un dispositivo di comunicazione (COM1: o COM2:), LOF restituisce il numero di caratteri disponibili in quel buffer di output. Questa funzione non può essere utilizzata con un numero associato ai dispositivi SCRN:, KYBD:, CONS: o LPTn.

Esempio

Il programma di seguito riportato utilizza la funzione LOF per verificare le dimensioni di un file.

```
CLS

OPEN "EX-LOF.DAT" FOR OUTPUT AS #1

FOR dimensioni% = 1 TO 23
    PRINT "La dimensione del file corrente è "; LOF(1); " byte."
    varStringa$ = STRING$(dimensioni%, "*") 'assegna asterischi alla variabile
    PRINT #1, varStringa$                    'scrive la variabile nel file
NEXT

CLOSE #1
```

LSET

Si consulti anche

FIELD, MKD$, MKDMBF$, MKI$, MKL$, MKS$, MKSMBF$, PUT (I/O su file), RSET

Scopo

L'istruzione LSET trasferisce il valore di un'espressione o di una variabile nel campo specificato di un buffer di un file ad accesso casuale, allineandolo a sinistra; può anche essere utilizzata per giustificare a sinistra qualsiasi variabile stringa, oppure per assegnare una variabile record a un'altra.

Sintassi

```
LSET var = espressione
```

var ed espressione devono essere entrambe stringhe, se LSET viene utilizzata per giustificare a sinistra o per inserire un valore in un campo; mentre devono essere entrambe variabili record, se LSET viene sfruttata per assegnare una variabile record a un'altra.

Utilizzo

```
LSET indirizzo$ = "Via Teano 9"
```

Giustifica a sinistra la costante stringa "Via Teano 9" e la inserisce nel campo indirizzo$, se quest'ultimo è il nome del campo definito dall'istruzione FIELD; se invece indirizzo$ è una variabile stringa, LSET si limita a giustificarla all'interno della lunghezza attuale di indirizzo$.

```
LSET ImpDov$ = MKD$(totale#)
```

Inserisce la rappresentazione di stringa del contenuto di totale# in ImpDov$, che può essere il nome di un campo definito da un'istruzione FIELD oppure una variabile stringa.

Descrizione

L'istruzione FIELD alloca un buffer speciale di memorizzazione, quando definisce il record per un file ad accesso casuale. A causa di questa speciale allocazione, non è possibile utilizzare il segno di uguale (=) o l'istruzione LET per assegnare i valori ai campi definiti dall'istruzione FIELD, ma è invece necessario sfruttare le istruzioni LSET o RSET. LSET può anche essere utilizzata con le variabili stringa e con le variabili record.

Utilizzo di LSET con i campi

Quando un nome di campo viene specificato come primo parametro in un'istruzione LSET, quest'ultima copia il valore del secondo parametro (che deve essere una stringa) nel buffer ad accesso casuale allocato per il campo specificato.

Se la lunghezza del campo è superiore a quella del valore della stringa, LSET giustifica a sinistra il valore nel buffer, ovvero aggiunge alcuni spazi a destra della stringa, fino a quando quest'ultima ha la stessa lunghezza del campo.

Se la lunghezza del campo è inferiore a quella del valore della stringa, LSET copia soltanto il numero di caratteri sufficienti a riempire il campo e ignora quelli restanti.

Nella Figura 17.5 viene illustrato il modo in cui l'istruzione LSET inserisce una stringa in un campo.

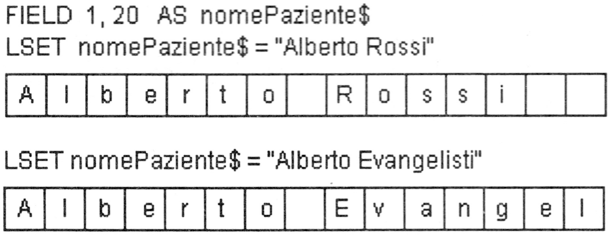

Figura 17.5 L'istruzione LSET con i campi

Utilizzo di LSET con le variabili stringa

Quando il primo parametro di LSET è una variabile stringa (non un nome di campo), LSET esegue un'operazione diversa. Dato che la lunghezza della variabile non è impostata da un'istruzione FIELD, LSET utilizza la lunghezza attuale della variabile stringa per calcolare la giustificazione da eseguire. Se questo valore è superiore alla lunghezza del valore specificato nel secondo parametro, LSET lo giustifica a sinistra, ovvero aggiunge alcuni spazi a destra della stringa, fino a quando quest'ultima ha la stessa lunghezza della variabile stringa.

Se la lunghezza della variabile stringa è inferiore a quella del valore indicato nel secondo parametro, LSET copia soltanto il numero di caratteri sufficienti a riempire la variabile stringa e ignora quelli restanti.

Per determinare la lunghezza attuale di una variabile stringa, è sufficiente utilizzare la funzione LEN. Nella Figura 17.6 viene illustrato il modo in cui l'istruzione LSET opera con le variabili stringa.

Utilizzo di LSET con le variabili record

Le funzioni consuete di assegnazione (il segno di uguale o LET) non consentono di copiare le variabili record definite da diverse istruzioni TYPE...END TYPE. Per copiare variabili record diverse è necessario utilizzare l'istruzione LSET, tuttavia questa istruzione si limita a copiare da una variabile ad un'altra, senza giustificare a sinistra i dati.

Quando LSET viene utilizzata in questo modo, i due parametri devono essere entrambi variabili record. L'istruzione copia i byte dalla variabile record specificata nel secondo parametro nella variabile record indicata nel primo parametro. Ad esempio, se la lunghezza della seconda variabile record è di 40 byte e la lunghezza della prima è di 30 byte, LSET copia soltanto 30 byte dalla seconda nella prima. Dato che l'istruzione LSET copia i singoli byte nella variabile record, i valori degli elementi della variabile non vengono copiati accuratamente, a meno che gli elementi della variabile in cui vengono copiati i byte non abbiano lo stesso tipo e posizione di quella che hanno nella seconda.

Il programma di seguito riportato illustra l'utilizzo di LSET con le variabili record.

```
FIELD 1, 20 AS nomePaziente$
LSET nomePaziente$ = "Alberto Rossi"
```

A	l	b	e	r	t	o		R	o	s	s	i		

```
LSET nomePaziente$ = "Alberto Evangelisti"
```

A	l	b	e	r	t	o		E	v	a	n	g	e	l

Figura 17.6 L'istruzione LSET con le variabili stringa

```
TYPE Tipo1
    x AS INTEGER
    y AS LONG
    z AS DOUBLE
END TYPE

TYPE Tipo2
    x AS INTEGER
    y AS LONG
END TYPE

DIM a1 AS Tipo1, a2 AS Tipo2

a1.x = 100
a1.y = 10000
a1.z = 100.1

a2.x = 200
a2.y = 200000

LSET a1 = a2
```

Dopo l'esecuzione dell'istruzione LSET, a1.x è uguale a 200, a1.y è uguale a 200000 e a1.z è uguale a 100.1. Dato che la variabile record a2 ha un numero inferiore di byte rispetto ad a1, il valore di a1.z rimane invariato.

Commenti

È necessario tenere presente che i valori numerici devono essere convertiti in stringhe, prima di inserirli in un campo definito dall'istruzione FIELD.

Esempio

Si consulti la voce CVD per esaminare un programma che utilizza l'istruzione LSET per inserire le stringhe nei campi che vengono memorizzati in un file ad accesso casuale.

LTRIM$

Si consulti anche

RTRIM$

Scopo

La funzione LTRIM$ restituisce una copia di un'espressione stringa senza spazi iniziali; può essere utilizzata per formattare i dati, nel caso in cui gli spazi iniziali potrebbero causare un allineamento sbagliato o essere erroneamente interpretati come caratteri letterali.

Sintassi

```
LTRIM$(stringa)
```

stringa è una qualsiasi espressione stringa.

Utilizzo

```
PRINT #1, LTRIM$(descrizione$)
```

Scrive nel file associato al numero 1 la variabile descrizione$, senza spazi iniziali.

Descrizione

QBasic fornisce la funzione LTRIM$ per permettere di cancellare gli spazi aggiuntivi che potrebbero essere stati inseriti da un utente o dal programma. Questa funzione accetta qualsiasi espressione stringa e restituisce la stessa stringa senza spazi iniziali.

Esempio

Il programma di seguito riportato utilizza le funzioni LTRIM$ e RTRIM$ per cancellare gli spazi iniziali e finali dalle righe di un file di testo.

```
CLS

PRINT "Questo programma elimina gli spazi iniziali e finali da ogni riga di "
PRINT "un file e scrive le righe risultanti su un altro file."
PRINT

LINE INPUT "Immettere il nome del file di input: "; nomeFile$
LINE INPUT "Immettere il nome del file di output: "; output$

OPEN nomeFile$ FOR INPUT AS #1
OPEN output$ FOR OUTPUT AS #2

DO UNTIL EOF(1)
   LINE INPUT #1, lineaInput$
   PRINT #1, LTRIM$(RTRIM$(lineaInput$))
LOOP

CLOSE
```

MKD$

Si consulti anche

CVD, CVDMBF, DOUBLE, FIELD, MKDMBF$

Scopo

La funzione MKD$ converte un numero a precisione doppia in una stringa di 8 byte; è il complemento di CVD e deve essere utilizzata per convertire un numero a precisione doppia in una stringa che viene assegnata al campo definito dall'istruzione FIELD.

Sintassi

```
MKD$(numero)
```

numero è una qualsiasi espressione numerica nell'intervallo da -1.7976931D+308 a 1.7976931D+308.

Utilizzo

```
LSET ImportoDovuto$ = MKD$(265.98)
```

Inserisce la stringa rappresentante il valore a precisione doppia 265.98 nel campo ImportoDovuto$.

Descrizione

La funzione MKD$ converte un valore a precisione doppia in una stringa di 8 byte. Si noti che il numero specificato può essere una qualsiasi espressione numerica; MKD$ la converte in un valore a precisione doppia. Il valore restituito può essere assegnato a una variabile oppure utilizzato in un'espressione; generalmente, MKD$ viene utilizzata in un'istruzione LSET.

Commenti

Se viene specificato il parametro /MBF dalla riga di comando, QBasic presuppone che il valore convertito da MKD$ sia un numero in formato binario Microsoft. In caso contrario, questa funzione converte un numero nel formato IEEE.

Esempio

Si consulti la voce CVD.

MKDMBF$

Si consulti anche

CVD, CVDMBF, DOUBLE, FIELD, MKD$

Scopo

La funzione MKDMBF$ converte un valore in precisione doppia nel formato IEEE in una stringa di 8 byte nel formato numerico binario Microsoft; è il complemento di CVDMBF e deve essere utilizzata per convertire un numero a precisione doppia in una stringa che viene assegnata al campo definito dall'istruzione FIELD.

Sintassi

```
MKDMBF$(numero)
```

numero è una qualsiasi espressione numerica nell'intervallo da -1.7976931D+308 a 1.7976931D+308.

Utilizzo

```
LSET ImportoDovuto$ = MKDMBF$(265.98)
```

Inserisce la stringa rappresentante il valore a precisione doppia 265.98 nel campo ImportoDovuto$.

Descrizione

La funzione MKD$ converte un valore a precisione doppia nel formato IEEE in una stringa di 8 byte nel formato binario Microsoft. Si noti che il numero specificato può essere una qualsiasi espressione numerica; MKDMBF$ la converte in un valore a precisione doppia. Il valore restituito può essere assegnato a una variabile oppure utilizzato in un'espressione; generalmente, MKDMBF$ viene utilizzata in un'istruzione LSET.

Esempio

Si consulti la voce CVD.

MKI$

Si consulti anche

CVS, CVSMBF, FIELD, MKSMBF$, SINGLE

Scopo

La funzione MKI$ converte un numero intero in una stringa di 2 byte; è il complemento di CVI e deve essere utilizzata per convertire un numero intero in una stringa che viene assegnata al campo definito dall'istruzione FIELD.

Sintassi

```
MKI$(numero)
```

numero è una qualsiasi espressione numerica nell'intervallo da -32768 a 32767.

Utilizzo

```
LSET omni$ = MKI$(19)
```

Inserisce la stringa rappresentante il valore intero 19 nel campo omni$.

Descrizione

La funzione MKI$ converte un valore intero in una stringa di 2 byte. Si noti che il numero specificato può essere una qualsiasi espressione numerica nell'intervallo

degli interi (da -32768 a 32767); MKI$ la converte in un intero. Il valore restituito può essere assegnato a una variabile oppure utilizzato in un'espressione; generalmente, MKI$ viene utilizzata in un'istruzione LSET$.

Esempio

Si consulti la voce CVD.

MKL$

Si consulti anche

CVL, FIELD, LONG

Scopo

La funzione MKL$ converte un numero intero lungo in una stringa di 4 byte; è il complemento di CVL e deve essere utilizzata per convertire un numero intero lungo in una stringa che viene assegnata al campo definito dall'istruzione FIELD.

Sintassi

```
MKL$(numero)
```

numero è una qualsiasi espressione numerica nell'intervallo da -2147483648 a 2147483647.

Utilizzo

```
LSET popolazione$ = MKL$(225102963)
```

Inserisce la stringa rappresentante il valore intero lungo 225102963 nella variabile popolazione$.

Descrizione

La funzione MKL$ converte un valore intero lungo in una stringa di 4 byte. Si noti che il numero specificato può essere una qualsiasi espressione numerica nell'intervallo degli interi lunghi (da -2147483648 a 2147483647); MKL$ la converte in un intero lungo. Il valore restituito può essere assegnato a una variabile oppure utilizzato in un'espressione; generalmente, MKL$ viene utilizzata in un'istruzione LSET$.

Esempio

Si consulti la voce CVD.

MKS$

Si consulti anche

CVS, CVSMBF, FIELD, MKSMBF$, SINGLE

Scopo

La funzione MKS$ converte un numero a precisione semplice in una stringa di 4 byte; è il complemento di CVS e deve essere utilizzata per convertire un numero a precisione semplice in una stringa che viene assegnata al campo definito dall'istruzione FIELD.

Sintassi

```
MKS$(numero)
```

numero è una qualsiasi espressione numerica nell'intervallo da -3.402823E+38 a 3.402823E+38.

Utilizzo

```
LSET venditeGlobali$ = MKS$(NostriCosti!)
```

Inserisce la stringa rappresentante il valore NostriCosti! nel campo venditeGlobali$.

Descrizione

La funzione MKS$ converte un valore a precisione semplice in una stringa di 4 byte. Si noti che il numero specificato può essere una qualsiasi espressione numerica; MKS$ la converte in un valore a precisione semplice. Il valore restituito può essere assegnato a una variabile oppure utilizzato in un'espressione; generalmente, MKS$ viene utilizzata in un'istruzione LSET.

Commenti

Se viene specificato il parametro /MBF dalla riga di comando, QBasic presuppone che il valore convertito da MKS$ sia un numero in formato binario Microsoft. In caso contrario, questa funzione converte un numero nel formato IEEE.

Esempio

Si consulti la voce CVD.

MKSMBF$

Si consulti anche

CVS, CVSMBF, FIELD, MKS$, SINGLE

Scopo

La funzione MKSMBF$ converte un valore in precisione semplice nel formato IEEE in una stringa di 4 byte nel formato numerico binario Microsoft; è il complemento di CVSMBF e deve essere utilizzata per convertire un numero a precisione semplice in una stringa che viene assegnata al campo definito dall'istruzione FIELD.

Sintassi

```
MKSMBF$(numero)
```

numero è una qualsiasi espressione numerica nell'intervallo da -3.402823E+38 a 3.402823E+38.

Utilizzo

```
LSET ImportoDovuto$ = MKSMBF$(265.98)
```

Inserisce la stringa rappresentante il valore a precisione semplice 265.98 nel campo ImportoDovuto$.

Descrizione

La funzione MKSMBF$ converte un valore a precisione semplice nel formato IEEE in una stringa di 4 byte nel formato binario Microsoft. Si noti che il numero specificato può essere una qualsiasi espressione numerica; MKSMBF$ la converte in un valore a precisione semplice. Il valore restituito può essere assegnato a una variabile oppure utilizzato in un'espressione; generalmente, MKSMBF$ viene utilizzata in un'istruzione LSET.

Esempio

Si consulti la voce CVD.

NAME

Si consulti anche

CHDIR

Scopo

L'istruzione NAME modifica il nome o la posizione di un file; può essere utilizzata per consentire al programma di gestire in modo efficace i file.

Sintassi

```
NAME vecchionomefile AS nuovonomefile
```

vecchionomefile è il nome del file esistente e può comprendere anche il percorso.
nuovonomefile è il nuovo nome da assegnare al file e può comprendere anche il percorso.

Utilizzo

```
NAME "CLIENTE.DAT" AS "CLIPREC.DAT"
```

Rinomina il file CLIENTE.DAT in CLIPREC.DAT.

```
NAME "\CONFIG.SYS" AS "\BIN\CONFIG.BAK"
```

Sposta il file CONFIG.SYS dalla vecchia directory nella directory \BIN e modifica il nome del file in CONFIG.BAK.

Descrizione

In modo analogo al comando RENAME del DOS, l'istruzione NAME modifica il nome di un file su disco e consente inoltre di spostarlo da una directory a un'altra sulla stessa unità. Tuttavia, NAME non è in grado di rinominare le directory.

I due parametri nella sintassi di NAME devono essere costanti stringa o espressioni che contengono un nome di file, che può essere preceduto da un percorso. Se il percorso del nuovo nome è diverso da quello del nome precedente, NAME sposta il file nella directory specificata e, se il nome del file è diverso, lo rinomina.

Errori

L'istruzione NAME non trasferisce fisicamente i file di dati, ma si limita a modificare le informazioni riguardanti il file nella directory del DOS. Se i due parametri non utilizzano la stessa unità QBasic visualizza il messaggio di errore: "Riassegnazione di nome tra unità disco diverse".

Se il file da rinominare non esiste, QBasic visualizza il messaggio di errore: "File non trovato"; se invece il nuovo nome del file esiste già, QBasic visualizza il messaggio di errore: "File già esistente".

Prima di utilizzare questa istruzione per rinominare o spostare un file, è necessario chiudere il file aperto, in caso contrario QBasic visualizza il messaggio di errore: "File già aperto".

Esempio

Il programma di seguito riportato utilizza l'istruzione NAME per effettuare la copia di backup del file esistente, prima di sovrascriverlo con i nuovi dati.

```
CLS
ON ERROR GOTO GestErrore  'passa  il  controllo  a GestErrore in caso di errore

NAME "RAPPORTO.RPT" AS "RAPPORTO.BAK"          'se il file esiste lo rinomina

OPEN "RAPPORTO.RPT" FOR OUTPUT AS #1           'apre il file per l'output

PRINT #1, "RAPPORTO.RPT"
PRINT "L'informazione è stata scritta nel file RAPPORTO.RPT"
CLOSE
END

GestErrore:
    SELECT CASE ERR
        CASE 53
             PRINT "RAPPORTO.RPT non trovato. Nessun backup è stato fatto."
             RESUME NEXT
        CASE 58
             KILL "RAPPORTO.BAK"
             PRINT "Il precedente file di backup è stato sostituito da uno
             nuovo."
             RESUME
        CASE ELSE
             ON ERROR GOTO 0
    END SELECT
```

OPEN

Si consulti anche

CLOSE, FREEFILE, IOCTL, OPEN COM

Scopo

L'istruzione OPEN apre un file o un dispositivo, in modo che il programma possa eseguire operazioni di input e di output; prepara i file che devono essere letti o scritti e deve essere utilizzata prima di usufruire delle istruzioni di I/O, quali GET, PUT, PRINT # o INPUT #.

Sintassi

Nuova sintassi:

```
OPEN file [FOR mod] [ACCESS accesso] [blocco] AS [#]numfile [LEN=lungrec]
```

Sintassi tradizionale:

```
OPEN modalità,[#]numfile,file[,lungrec]
```

file è il nome del file da aprire.

mod è una parola chiave che indica una tra le seguenti modalità di apertura dei file: APPEND, BINARY, INPUT, OUTPUT o RANDOM.

accesso è la parola chiave che specifica il tipo di accesso che il programma adotta per il file in fase di apertura e può essere una delle seguenti: READ, WRITE o READ WRITE.

blocco è una clausola che indica il tipo di blocco del file e può essere una delle seguenti: SHARED, LOCK READ, LOCK WRITE, LOCK READ WRITE.

numfile è un numero compreso tra 1 e 255 che identifica un file aperto e può essere utilizzato per accedere al file.

lungrec è la lunghezza di ogni record per i file ad accesso casuale o per quelli sequenziali e deve trovarsi nell'intervallo da 1 a 32767.

modalità è una singola lettera che indica il modo in cui il programma utilizza i file; può essere una delle seguenti: I, O, A, R o B.

Utilizzo

```
OPEN "A:MODULI.DAT" FOR INPUT AS #1
```

Apre il file sequenziale MODULI.DAT sull'unità A: come numero 1 e indica i dati da leggere.

```
OPEN "D:\DEGENTI.DAT" FOR RANDOM ACCESS READ WRITE LOCK WRITE AS #3 LEN -1024
```

Apre il file ad accesso casuale DEGENTI.DAT sull'unità D. L'operazione corrente può leggere o scrivere nel file, ma le altre operazioni non possono aprire il file per la scrittura. In questo caso il file è stato aperto come numero 3 e ogni record ha una lunghezza di 1024 byte.

Descrizione

Prima di poter eseguire le operazioni di lettura e scrittura di un file, è necessario aprirlo. L'istruzione OPEN è in grado di comunicare con il DOS e con la rete locale per impostare la modalità di accesso e il bloccaggio del file, consentendo in questo modo a diversi programmi QBasic di accedere agli stessi file.

Il file da aprire deve essere specificato da una costante stringa o espressione che rappresenta un file o un dispositivo del DOS o di QBasic. Si noti che il nome del file può essere preceduto dal nome dell'unità disco, dal nome della directory o da entrambi, in base alle convenzioni DOS. Nella Tabella 17.5 è riportato un elenco dei nomi di dispositivo corretti di QBasic.

La clausola LEN, che è facoltativa, consente di specificare la lunghezza in byte dei record presenti nel file o le dimensioni, sempre in byte, del buffer di QBasic relativo a quel file. Per quanto riguarda i file ad accesso casuale, la clausola LEN indica il numero di byte in ogni record del file; se viene omessa, la lunghezza di default è di 128 byte. Questa clausola viene ignorata per i file binari. Nel caso dei file sequenziali, la clausola LEN specifica le dimensioni (in byte) del buffer utilizzato da QBasic durante le operazioni di scrittura e di lettura; il valore di default è di 512 byte. Specificando un buffer di ampie dimensioni le operazioni di input e output vengono accelerate, tuttavia QBasic alloca un numero di byte leggermente inferiore rispetto a quello specificato dalla memoria disponibile, rendendo quindi necessario decidere se un aumento della velocità valga la memoria aggiuntiva che il buffer richiede. Se il file è un dispositivo a caratteri, QBasic ignora la lunghezza dato che, in questo caso, il buffer è sempre di 1 byte e non può essere modificato.

La clausola FOR

Nella nuova sintassi di OPEN, la modalità di accesso viene specificata utilizzando la clausola FOR con una delle seguenti parole chiave: APPEND, BINARY, INPUT, OUTPUT o RANDOM.

Nella sintassi tradizionale invece, la modalità di accesso viene specificata inserendo la prima lettera di una delle parole chiave della nuova sintassi (I, O, A, R o B). Utilizzando la sintassi tradizionale, è necessario inserire la modalità di accesso.

Nella Tabella 17.6 sono riportate le definizioni relative alle modalità di accesso. Se la clausola FOR viene omessa nella nuova sintassi, QBasic apre il file utilizzando la modalità ad accesso casuale.

Quando un file viene aperto con la parola chiave BINARY nella clausola FOR, le variabili possono essere lette e scritte in qualsiasi posizione, indipendentemente dalla struttura interna del file.

Se il file consiste di testo ASCII immesso su righe separate oppure separato da spazi o virgole, è necessario utilizzare la parola chiave INPUT nella clausola FOR per aprire il file in formato sequenziale. Nella modalità input, i dati presenti nel file possono essere privi di struttura e di tipi diversi (numerici o di testo), dato che le routine di I/O INPUT # e LINE INPUT # leggono il file ed effettuano le conversioni necessarie (ad esempio da caratteri ASCII a variabili numeriche).

Per creare un file sequenziale, è sufficiente specificare la parola chiave OUTPUT nella clausola FOR. I dati scritti in un file aperto per l'output possono essere privi di struttura e di tipi diversi.

Tabella 17.5 I dispositivi di QBasic

Dispositivo	**Descrizione**
CONS:	Console; output sullo schermo e input da tastiera
KYBD:	Tastiera; soltanto input da tastiera
SCRN:	Schermo; soltanto output sullo schermo
COMn:	Porta di comunicazione n; input e output
LPTn:	Porta parallela della stampante; input e output

Tabella 17.6 Le modalità di accesso specificate dalla clausola FOR

Nuova sintassi	**Sintassi tradizionale**	**Descrizione**
INPUT	I	Input sequenziale
OUTPUT	O	Output sequenziale
APPEND	A	Output sequenziale fino alla fine del file
RANDOM	R	I/O ad accesso casuale (modalità di default) in qualsiasi posizione del record
BINARY	B	I/O binario in qualsiasi posizione del byte

Per aprire un file sequenziale, a cui è necessario aggiungere alcune informazioni, è necessario specificare APPEND. In questo caso, i dati scritti nel file appaiono alla fine, mentre i dati originali rimangono invariati.

Specificando RANDOM in una clausola FOR, è possibile aprire un file ad accesso casuale che memorizza record di dati che hanno la stessa struttura. Questi record possono essere definiti da un'istruzione FIELD o da TYPE...END TYPE. In questi tipi di file è possibile accedere direttamente a qualsiasi record, senza dover iniziare la ricerca all'inizio del file.

La clausola ACCESS

La clausola ACCESS può comprendere una delle seguenti parole chiave: READ, WRITE o READ WRITE, che indicano che il file deve essere aperto soltanto per la lettura, soltanto per la scrittura, per entrambe le operazioni. ACCESS READ WRITE è valida solo per i file aperti con le parole chiave RANDOM, BINARY o APPEND nella clausola FOR.

Se il processo che apre il file nega l'accesso per la lettura, qualsiasi tentativo di aprire quel file con ACCESS READ o ACCESS READ WRITE provoca la visualizzazione del messaggio di errore: "Permesso negato". La stessa situazione si verifica se viene aperto un file che non contempla l'accesso per le operazioni di scrittura.

Se non viene specificata la clausola ACCESS con un file ad accesso casuale o in formato binario, QBasic effettua tre tentativi per aprire il file, uno con una clausola

implicita ACCESS READ WRITE, il successivo con un ACCESS WRITE implicito e l'ultimo con un ACCESS READ implicito. Nel caso di un conflitto per la condivisione del file, viene effettuato un tentativo successivo. Se nessuno di questi ha successo, QBasic visualizza il messaggio di errore: "Permesso negato".

La clausola blocco

La clausola blocco indica al sistema operativo quali diritti di accesso possono essere utilizzati per quel file. Ad esempio, se la clausola è LOCK READ, nessun altro processo può accedere al file con i diritti di lettura. Nella Tabella 17.7 sono riportate le parole chiave che possono essere specificate in una clausola blocco.

Errori

Se l'istruzione OPEN viene utilizzata con la parola chiave INPUT (o I) e il file non esiste, QBasic visualizza il messaggio di errore: "File non trovato". In qualsiasi altra modalità invece, QBasic si limita a creare il file con una lunghezza di 0 byte, ma non visualizza alcun messaggio.

Suggerimenti

Dato che ogni buffer di file aperto utilizza la memoria disponibile, è opportuno chiudere un file, quando non è più necessario utilizzarlo.

Avvertenze

Se viene utilizzata una funzione definita dall'utente o una funzione DEF FN come parametro per un'istruzione di I/O, quella funzione non deve essa stessa utilizzare questo tipo di istruzione, dato che in tal caso i risultati sarebbero imprevedibili.

Quando un file viene aperto con la parola chiave APPEND, la clausola ACCESS, se utilizzata, deve essere READ WRITE e non semplicemente WRITE, dato che l'istruzione OPEN deve avere i diritti di lettura per essere in grado di individuare la fine del file.

Tabella 17.7 Le definizioni della modalità di accesso in rete

Clausola	Descrizione
(Default)	Nessun'altra elaborazione può aprire il file.
SHARED	Qualsiasi elaborazione può aprire il file in qualsiasi modalità.
LOCK READ	Nessun'elaborazione può aprire il file con la clausola ACCESS READ; nessun'altra elaborazione può già avere i diritti ACCESS WRITE.
LOCK WRITE	nessuna elaborazione può aprire i file con la clausola ACCESS WRITE; nessun'altra elaborazione può già avere i diritti ACCESS WRITE.
LOCK READ WRITE	Nessuna elaborazione può aprire il file con la clausola ACCESS READ WRITE; nessun'altra elaborazione può già avere i diritti ACCESS READ WRTE.

Esempio

L'esempio di seguito riportato, che utilizza tre istruzioni OPEN, converte un file di testo nei formati esadecimale e ottale.

```
OPEN "INPUT.DAT" FOR INPUT AS #1
OPEN "HEX.DAT" FOR OUTPUT AS #2
OPEN "OTTALE.DAT" FOR OUTPUT AS #3

DO UNTIL EOF(1)
   LINE INPUT #1, testo$                      'legge una riga
   PRINT testo$

   PRINT #2, testo$
   FOR x = 1 TO LEN(testo$)
       PRINT #2, HEX$(ASC(MID$(testo$, x, 1))); " ";     'stampa il valore
                                                          'esadecimale
   NEXT
   PRINT #2,

   PRINT #3, testo$
   FOR x = 1 TO LEN(testo$)
       PRINT #3, OCT$(ASC(MID$(testo$, x, 1))); " "; 'stampa il valore ottale
   NEXT
   PRINT #3,
LOOP

CLOSE                                         'chiude tutti i file aperti
```

PRINT

Si consulti anche

INPUT #, PRINT, PRINT # USING, PRINT USING, WRITE #

Scopo

L'istruzione PRINT # scrive dati non formattati in un file sequenziale; può esser utilizzata per aggiungere testo e numeri a un file di questo tipo.

Sintassi

```
PRINT #numfile[,espressioni[{; | ,}espressioni...]][; | ,]
```

numfile è il numero del file sequenziale aperto, in cui vengono scritti i dati.

espressioni è un'espressione numerica o stringa che contiene il valore da scrivere nel file; è possibile specificare diverse espressioni, separandole con punti e virgola o virgole.

Se tra due espressioni si utilizza un punto e virgola, la seconda viene scritta subito dopo la prima; se invece viene inserita una virgola, la seconda espressione viene scritta nella successiva zona di stampa.

Un punto e virgola o una virgola alla fine dell'istruzione mantengono il puntatore sulla stessa riga del file, in modo tale che l'output della successiva istruzione PRINT # inizi sulla stessa riga.

Utilizzo

```
PRINT #5, "La data di oggi è "; DATE$
```

Scrive una costante stringa e il valore restituito dalla funzione DATE$ nel file aperto come numero 5.

Descrizione

L'output dei file sequenziali risulta utile nella creazione di file che gli utenti possono visualizzare o modificare, quali quelli relativi ai prospetti o alla fusione di due archivi, di cui uno contiene il testo e l'altro nomi e indirizzi. Si noti che il file in cui vengono scritti i dati deve essere stato in precedenza aperto utilizzando la parola chiave OUTPUT o APPEND.

Il valore scritto nel file può essere specificato da una costante (testo tra apici o una costante definita da CONST), oppure da un'espressione di qualsiasi tipo semplice, quale un elemento di array, un elemento record, una variabile semplice oppure un'espressione stringa o matematica che utilizzi le funzioni di QBasic. Se non viene specificato alcun valore, PRINT # scrive una riga vuota.

Questa istruzione scrive i dati nel file, proprio come PRINT visualizza i dati sullo schermo. Se è necessario leggere un file in un momento successivo, occorre inserire tra le voci alcuni delimitatori (quali una coppia ritorno a capo/avanzamento riga, spazi oppure virgole).

Scrittura dei valori numerici

Se un parametro di PRINT # è una variabile numerica o un'espressione, l'istruzione ne scrive il valore nel file. Quando PRINT # scrive i numeri in un file, inserisce uno spazio dopo ogni numero, facendo precedere i valori numerici positivi da uno spazio e quelli negativi da un segno meno.

Se il parametro è una variabile di tipo intero, intero lungo o espressione, PRINT # ne scrive i valori nel file utilizzando le cifre necessarie a rappresentare il numero. Tranne che per i casi indicati in precedenza, il valore non viene preceduto da nessuno spazio iniziale o zero, quando viene scritto nel file. Nella Figura 17.7 viene illustrata questa procedura.

Se un parametro è una variabile a precisione semplice o a precisione doppia, PRINT # ne scrive il valore nel file utilizzando le cifre necessarie a rappresentare il numero. Se QBasic può rappresentare il valore di un numero a precisione semplice in sette cifre o meno (più il punto decimale), senza andare a scapito della precisione, il valore viene scritto nel file in un formato a virgola fissa; in caso contrario, questo valore viene scritto in notazione scientifica. Se QBasic è in grado di rappresentare il valore di un numero a doppia precisione in 16 cifre o meno (più il punto decimale), senza andare a scapito della precisione, il valore viene scritto nel file in un formato a virgola fissa; in caso contrario, questo valore viene scritto in notazione scientifica. Nella Figura 17.8 viene illustrata questa procedura.

Commenti

Se per separare i parametri vengono utilizzate le virgole, PRINT # inserisce degli spazi nel file in corrispondenza della successiva posizione di tabulazione. Dato che

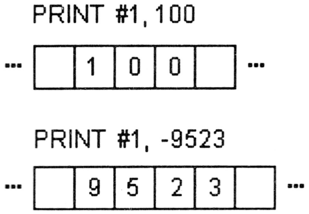

Figura 17.7 L'utilizzo di PRINT # per scrivere interi in un file

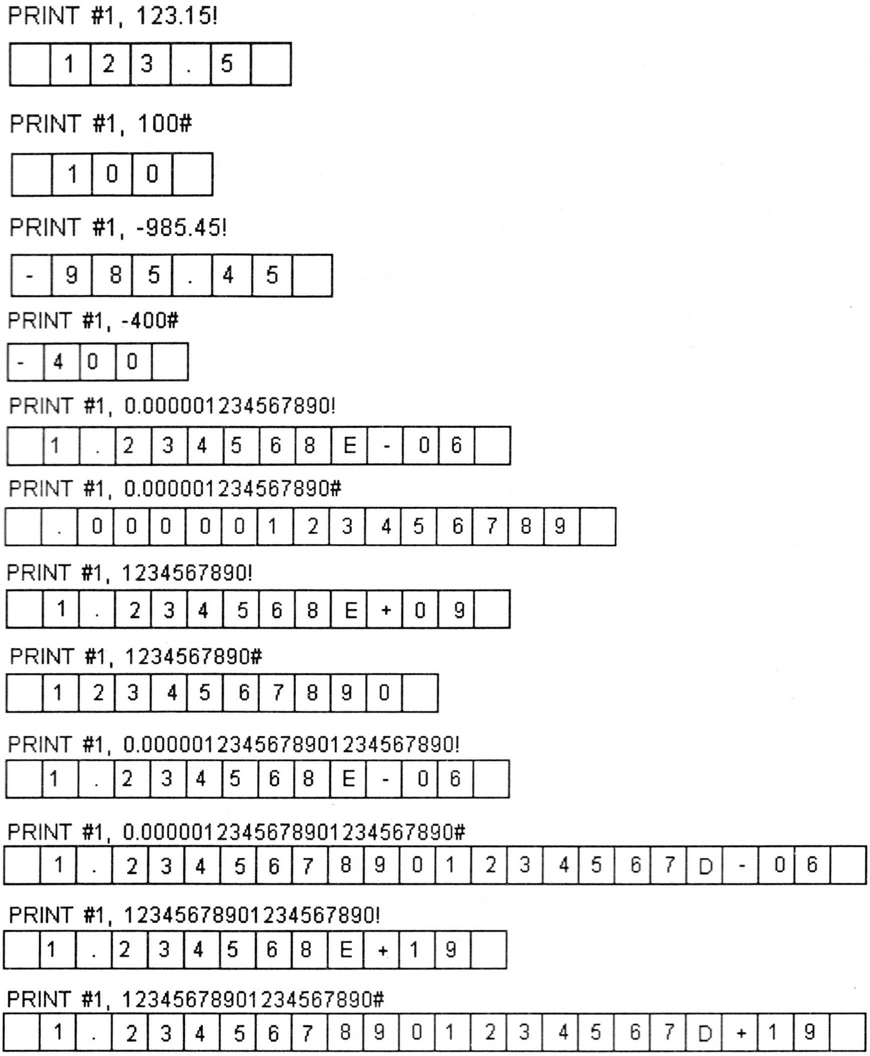

Figura 17.8 L'utilizzo di PRINT # per scrivere numeri a precisione semplice e a precisione doppia in un file

questo può ridurre lo spazio su disco, è meglio utilizzare i punti e virgola tra i valori numerici e i doppi apici tra le variabili stringa, come nel seguente esempio:

```
PRINT #1, Varstringa$; ","; VarNumerica%; VarLire#
```

Errori

Se come parametro viene utilizzato un array o una variabile record, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Suggerimenti

Per velocizzare l'output del file sequenziale è possibile aumentare le dimensioni del buffer, utilizzando la clausola LEN nell'istruzione OPEN. Tuttavia, è necessario tenere presente che le dimensioni maggiori del buffer riducono la memoria disponibile per il programma.

Esempio

Il programma di seguito riportato utilizza l'istruzione PRINT # per scrivere il contenuto di uno schermo di testo di 80 per 25 caratteri in un file sequenziale.

```
OPEN "SCHERMO.DAT" FOR APPEND AS #1

FOR y% = 1 TO 25
    FOR x% = 1 TO 80
        PRINT #1, CHR$(SCREEN(y%, x%));
    NEXT x%

NEXT y%

PRINT #1, STRING$(80, 196)
PRINT #1, "Questa schermata è stata realizzata il "; DATE$; " alle "; TIME$;
"."
PRINT #1, STRING$(80, 196)

CLOSE
```

PRINT # USING

Si consulti anche

INPUT #, PRINT, PRINT #, PRINT USING, WRITE #

Scopo

L'istruzione PRINT # USING scrive dati formattati in un file sequenziale, proprio come PRINT USING li visualizza sullo schermo; può esser utilizzata per scrivere testo e numeri in un file che, ad esempio, deve essere sfruttato per compilare una parte determinata di un modulo.

Sintassi

```
PRINT #numfile, USING espressionestringa; espressioni[;espressione...][; | ,]
```

numfile è il numero del file sequenziale aperto, in cui vengono scritti i dati.

espressionestringa è una costante stringa o espressione stringa che può contenere testo letterale, che viene stampato invariato, e uno o più caratteri di formattazione, che indicano in che modo le variabili e le espressioni che seguono devono essere stampate.

espressione è un'espressione numerica o stringa, che contiene il valore da scrivere nel file; è possibile specificarne diverse, separandole con punti e virgola.

Se alla fine dell'istruzione viene inserita una virgola, il puntatore rimane sulla stessa riga del file e si sposta nella successiva zona di stampa. Un punto e virgola alla fine dell'istruzione fa in modo che il puntatore rimanga sulla stessa riga, subito dopo il carattere scritto per ultimo.

Utilizzo

```
PRINT #1, USING "######"; numVar%
```

Stampa nel file numero 1 sei cifre del contenuto della variabile numVar%.

```
PRINT #1, USING "Pagare all'ordine di !. &"; Nome$, Cognome$
```

Scrive una stringa composta da un carattere e una stringa a lunghezza variabile nel file numero 1.

Descrizione

Questa istruzione scrive i dati nel file, proprio come PRINT USING li visualizza sullo schermo. Se è necessario in un momento successivo leggere un file sequenziale aperto con la parola chiave INPUT, occorre inserire tra le voci alcuni delimitatori.

Errori

Se come parametro viene utilizzato un array o una variabile record, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Suggerimenti

Per velocizzare l'output del file sequenziale è possibile aumentare le dimensioni del buffer, utilizzando la clausola LEN nell'istruzione OPEN. Tuttavia, è necessario tenere presente che le dimensioni maggiori del buffer riducono la memoria disponibile per il programma.

Esempio

Il programma di seguito riportato utilizza l'istruzione PRINT # USING per scrivere i dati in file, che potrebbe successivamente essere stampato sugli assegni.

```
CLS
DEFINT A-Z              'definisce per default le variabili come intere

file = FREEFILE         'ottiene il successivo numero file disponibile
OPEN "ASSEGNI" FOR APPEND AS #file      'apre il file per output aggiuntivo

DO
   PRINT
   PRINT "Quando hai terminato, premi Invio."
   PRINT
```

```
    LINE INPUT "Immettere il nome del beneficiario: "; nome$
    IF nome$ = "" THEN EXIT DO          'esce se l'utente preme Invio
    LINE INPUT "Immettere il cognome del beneficiario: "; cognome$
    INPUT "Immettere l'importo da pagare: ", importo!

    PRINT #file, "Nuovo Istituto Bancario", , , DATE$    'le virgole
                                                         'specificano spazi di
                                                         'tabulazione
    PRINT #file, : PRINT #file, : PRINT #file,           'salta tre righe

    PRINT #file, "Pagare all'ordine di: "
    PRINT #file, USING "!.  \               \ "; nome$; cognome$;
    PRINT #file, USING "            L_. #,###,###"; importo!
    lire& = FIX(importo!)          'calcola il numero dell'importo complessivo
    PRINT #file, USING "#,###,### lire"; lire&
    PRINT #file, : PRINT #file, : PRINT #file,
    PRINT #file, , , "Il presidente Mario Rossi"
    PRINT #file, : PRINT #file, : PRINT #file,
LOOP

CLOSE #file
```

PUT (I/O su file)

Si consulti anche

FIELD, GET (I/O su file), INPUT #, LINE INPUT #, OPEN, TYPE...END TYPE

Scopo

L'istruzione PUT scrive in un file binario o ad accesso casuale le informazioni contenute nei campi definiti dall'istruzione FIELD o nelle variabili record definite da un'istruzione TYPE...END TYPE.

Sintassi

```
PUT[#]numfile[,[numerorecord][,var]]
```

numerofile è il numero di un file aperto.

numerorecord è il numero del record (nei file ad accesso casuale) o la posizione del byte (nei file in formato binario), in cui inizia la scrittura dei dati; è compreso nell'intervallo da 1 a 2147483647.

var è la variabile che contiene i dati da scrivere nel file.

Utilizzo

```
PUT #1, ,1
```

Scrive il contenuto del file associato a un file ad accesso casuale nel primo record del file aperto come numero 1.

```
PUT #1, , info
```

Scrive la variabile record info nel record successivo in un file ad accesso casuale.

Descrizione

L'istruzione PUT può essere utilizzata per scrivere semplici variabili o record completi in un file ad accesso casuale o binario.

Quando questa istruzione viene utilizzata con un file ad accesso casuale, è possibile specificare il numero del record in corrispondenza del quale inizia l'operazione di scrittura. Ogni record comprende un numero di byte specificati nella clausola LEN dell'istruzione OPEN. Il primo numero di record in un file è 1. Se non viene specificato alcun numero, PUT scrive sul record successivo a quello utilizzato nell'ultima operazione GET o PUT o dopo quello indicato nell'operazione SEEK più recente.

Nel caso dei file ad accesso casuale è possibile indicare qualsiasi variabile le cui dimensioni in byte non siano superiori alla lunghezza del record indicata quando il file è stato aperto. È necessario specificare una variabile che corrisponda alla struttura dei record dei file.

Quando PUT viene utilizzata con un file binario, è possibile specificare il numero del byte in cui inizia la scrittura. Il primo byte in un file è il numero 1; se non viene specificato alcun numero, PUT scrive a partire dal byte successivo a quello utilizzato nell'ultima operazione GET o PUT o a quello indicato nell'operazione SEEK più recente.

La variabile specificata per il parametro del file binario può essere di qualsiasi tipo e lunghezza, questa istruzione scrive il numero di byte contenuti in essa.

Se l'istruzione PUT viene utilizzata quando il record nel file è stato definito da un'istruzione FIELD, non è possibile utilizzare una variabile, altrimenti QBasic visualizza il messaggio di errore: "Istruzione FIELD attiva". Per scrivere i valori in un file, è necessario innanzitutto assegnare quei valori ai campi del record, utilizzando l'istruzione LSET o RSET, poi, è sufficiente utilizzare PUT senza una variabile per scrivere i valori nel file.

Avvertenze

Dato che la dimensione massima di un record in un file è di 32767, anche la dimensione massima della variabile in un'istruzione PUT per i file ad accesso casuale o per quelli binari è di 32767.

Se la variabile specificata è una stringa a lunghezza variabile, prima dei dati viene inserita un'intestazione di 2 byte, che indica la lunghezza della stringa. Quindi, ogni stringa amplia la lunghezza del record di due byte. Questo potrebbe far sorgere dei problemi, se i file di dati devono essere condivisi con un programma scritto con una versione precedente di QBasic; per impedire che ciò avvenga, è sufficiente utilizzare le stringhe a lunghezza fissa.

Esempio

Il programma di seguito riportato utilizza le istruzioni GET e PUT per aggiornare un file che contiene le scelte di colori effettuate dall'utente per quanto riguarda lo schermo.

```
CLS
TYPE TipoConfig
     normale AS INTEGER           'colore dell'area interna del riquadro
     altaInt AS INTEGER           'colore della lettera in alta intensità
     invAltaInt AS INTEGER        'colore dell'inverso della lettera in
                                  'alta intensità
     inverso AS INTEGER              'colore della barra in alta intensità
     aiuto AS INTEGER                'colore della riga di aiuto
     cursoreNorm AS INTEGER     'dimensione del cursore in modo sovrascrittura
```

```
    cursoreIns AS INTEGER      'dimensione del cursore in modo Inserimento
END TYPE
DIM config AS TipoConfig                'definisce la variabile record config

OPEN "CONFIG.DAT" FOR BINARY AS #1      'apre il file
GET #1, 1, config          'legge le informazioni sulla configurazione corrente

CLS
PRINT "Immettere i nuovi colori."
PRINT

PRINT "Colore normale = "; config.normale;
INPUT " Cambia in: ", config.normale

PRINT "Colore alta intensità = "; config.altaInt;
INPUT " Cambia in: ", config.altaInt

PRINT "Inverso colore alta intensità = "; config.invAltaInt;
INPUT " Cambia in: ", config.invAltaInt

PRINT "Inverso colore = "; config.inverso;
INPUT " Cambia in: ", config.inverso

PRINT "Colore aiuto = "; config.aiuto;
INPUT " Cambia in: ", config.aiuto

PRINT "Dimensione del cursore in modalità sovrascrittura = ";
config.cursoreNorm;
INPUT " Cambia in: ", config.cursoreNorm

PRINT "Dimensione del cursore in modalità inserimento = "; config.cursoreIns;
INPUT " Cambia in: ", config.cursoreIns

PUT #1, 1, config                       'scrive le informazioni nel file
CLOSE

END
```

RESET

Si consulti anche

CLOSE, END, SYSTEM

Scopo

L'istruzione RESET chiude tutti i file aperti e scrive i dati presenti nei buffer corrispondenti nel file o nel dispositivo corretti; può essere utilizzata alla fine del programma, per assicurarsi che tutti i file siano chiusi prima di terminare l'esecuzione.

Sintassi

```
RESET
```

Utilizzo

```
PRINT "Tutte le operazioni sono state completate."
RESET
```

Visualizza un messaggio di chiusura, cancella i dati dai buffer dei file in uso e quindi chiude tutti i file aperti.

Descrizione

L'istruzione RESET chiude tutti i file aperti e cancella i dati ancora presenti nei buffer che erano stati allocati dall'istruzione OPEN; funziona in modo analogo a CLOSE, senza alcun parametro.

Commenti

L'istruzione RESET (e CLOSE, END e SYSTEM) cancellano le operazioni di I/O pendenti, scrivendo il contenuto dei buffer in uso nel file o nel dispositivo corretto. Tuttavia, RESET non esegue un'istruzione PUT per aggiornare un file contenente campi (definiti dall'istruzione FIELD) in un file ad accesso casuale; se sono state apportate modifiche ai campi, è meglio eseguire un'istruzione PUT per aggiungere le modifiche al file, prima dell'istruzione RESET.

Esempio

Il programma di seguito riportato utilizza l'istruzione RESET per chiudere tutti i file, quando l'utente termina l'esecuzione.

```
OPEN "PRIMO" FOR INPUT AS #1
OPEN "SECONDO" FOR INPUT AS #2
OPEN "TERZO" FOR OUTPUT AS #3
OPEN "PIPPO.DAT" FOR APPEND AS #7

INPUT "Vuoi uscire dal programma ?"; siNo$

DO
  IF UCASE$(LEFT$(siNo$, 1)) = "S" THEN
     RESET
     END
  END IF
LOOP
```

RSET

Si consulti anche

FIELD, LSET, MKD$, MKDMBF$, MKI$, MKL$, MKS$, MKSMBF$, PUT (I/O su file)

Scopo

L'istruzione RSET trasferisce il valore di un'espressione o di una variabile nel campo specificato del buffer di un file ad accesso casuale, allineandolo a destra; può anche essere utilizzata per giustificare a destra qualsiasi variabile stringa.

Sintassi

```
RSET var = espressione
```

var ed espressione devono essere entrambe stringhe.

Utilizzo

```
RSET indirizzo$ = "Via Teano 9"
```

Giustifica a destra la costante stringa "Via Teano 9" e la inserisce nel campo indirizzo$, se quest'ultimo è il nome del campo definito dall'istruzione FIELD. Se invece indirizzo$ è una variabile stringa, RSET si limita a giustificarla all'interno della lunghezza attuale di indirizzo$.

```
RSET ImpDov$ = MKD$(totale#)
```

Inserisce la rappresentazione di stringa del contenuto di totale# in ImpDov$, che può essere il nome di un campo definito da un'istruzione FIELD oppure una variabile stringa.

Descrizione

L'istruzione FIELD alloca un buffer speciale di memorizzazione, quando definisce il record per un file ad accesso casuale. A causa di questa speciale allocazione, non è possibile utilizzare il segno di uguale (=) o l'istruzione LET per assegnare i valori ai campi definiti dall'istruzione FIELD, ma è invece necessario sfruttare le istruzioni RSET o RSET. RSET può anche essere utilizzata con le variabili stringa e con le variabili record.

Utilizzo di RSET con i campi

Quando un nome di campo viene specificato come il primo parametro in un'istruzione RSET, quest'ultima copia il valore del secondo parametro (che deve essere una stringa) nel buffer ad accesso casuale allocato per il campo specificato.

Se la lunghezza del campo è superiore a quella del valore della stringa, RSET giustifica a destra il valore nel buffer, ovvero aggiunge alcuni spazi all'inizio della stringa, fino a quando quest'ultima ha la stessa lunghezza del campo.

Se la lunghezza del campo è inferiore a quella del valore della stringa, RSET copia soltanto il numero di caratteri sufficienti a riempire il campo e ignora quelli restanti.

Nella Figura 17.9 viene illustrato il modo in cui l'istruzione RSET inserisce una stringa in un campo.

Utilizzo di RSET con le variabili stringa

Quando il primo parametro di RSET è una variabile stringa (non un nome di campo), RSET esegue un'operazione diversa. Dato che la lunghezza della variabile non è impostata da un'istruzione FIELD, RSET utilizza la lunghezza attuale della variabile stringa per calcolare la giustificazione da eseguire.

Se questo valore è superiore alla lunghezza del valore specificato nel secondo parametro, RSET lo giustifica a destra, ovvero aggiunge alcuni spazi all'inizio della stringa, fino a quando quest'ultima ha la stessa lunghezza della variabile stringa.

Se la lunghezza della variabile stringa è inferiore a quella del valore indicato nel secondo parametro, RSET copia soltanto il numero di caratteri sufficienti a riempire la variabile stringa e ignora quelli restanti. Per determinare la lunghezza attuale di una variabile stringa, è sufficiente utilizzare la funzione LEN.

Nella Figura 17.10 viene illustrato il modo in cui RSET gestisce le variabili stringa.

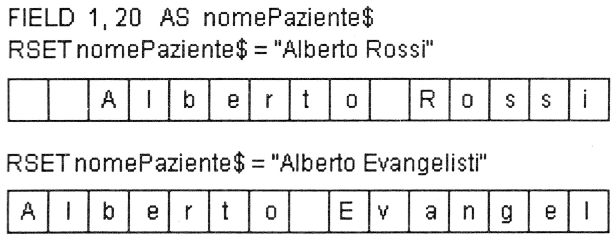

Figura 17.9 L'istruzione RSET con i campi

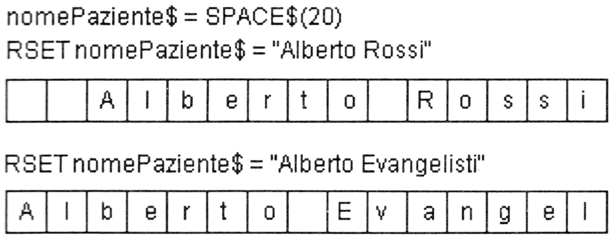

Figura 17.10 L'istruzione RSET con le variabili stringa

Commenti

È necessario tenere presente che i valori numerici devono essere convertiti in stringhe, prima di inserirli in un campo definito dall'istruzione FIELD.

Esempio

Il programma di seguito riportato utilizza RSET per giustificare a destra l'input dell'utente e scrive i titoli professionali e i nomi delle persone in un file, posizionando il titolo professionale nella parte sinistra e il nome corrispondente sulla destra. Quindi, visualizza il contenuto del file.

```
OPEN "CREDITI.TXT" FOR OUTPUT AS #1

DO
  lavoro$ = SPACE$(40)
  persona$ = SPACE$(40)

  LINE INPUT "Titolo professionale:   "; titoloProf$
  IF titoloProf$ = "" THEN EXIT DO
  LINE INPUT "Nome della persona: "; nomePersona$
  PRINT

  LSET lavoro$ = titoloProf$
  RSET persona$ = nomePersona$
  PRINT #1, lavoro$; persona$          'scrive l'informazione nel file
LOOP UNTIL titoloProf$ = ""

CLOSE #1

OPEN "CREDITI.TXT" FOR INPUT AS #1
```

```
DO UNTIL EOF(1)
   LINE INPUT #1, credito$                    'visualizza il contenuto del file
   PRINT credito$
LOOP
CLOSE #1
```

RTRIM$

Si consulti anche

LTRIM$

Scopo

La funzione RTRIM$ restituisce una copia di un'espressione stringa senza spazi finali; può essere utilizzata per formattare i dati, nel caso in cui gli spazi finali potrebbero causare un allineamento sbagliato o essere erroneamente interpretati come caratteri letterali.

Sintassi

```
RTRIM$(stringa)
```

stringa è una qualsiasi espressione stringa.

Utilizzo

```
PRINT #1, RTRIM$(descrizione$)
```

Scrive nel file associato al numero 1 la variabile descrizione$, senza spazi finali.

Descrizione

QBasic fornisce la funzione RTRIM$ per cancellare gli spazi aggiuntivi che potrebbero essere stati inseriti da un utente o dal programma. Questa funzione accetta qualsiasi espressione stringa e restituisce la stessa stringa, senza spazi finali.

Esempio

Il programma di seguito riportato utilizza la funzione RTRIM$ per cancellare gli spazi finali dall'input dell'utente.

```
CLS

LINE INPUT "Immettere il vostro nome: "; nome$
LINE INPUT "Immettere il vostro cognome: "; cognome$

PRINT
PRINT "Pagare all'ordine di: "; RTRIM$(nome$); " "; RTRIM$(cognome$)
```

SEEK (Funzione)

Si consulti anche

GET (I/O su file), OPEN, PUT (I/O su file), SEEK (Istruzione)

Scopo

La funzione SEEK restituisce il numero del record in uso o la posizione di un byte in un file; può essere utilizzata, ad esempio, per determinare quanto il record attivo sia vicino alla fine del file.

Sintassi

```
SEEK(numfile)
```

numfile è il numero del file aperto.

Utilizzo

```
PRINT "La posizione attuale del file è"; SEEK(1)
```

Visualizza la posizione attuale del file aperto come numero 1.

Descrizione

Il valore restituito da SEEK per i file ad accesso casuale è il numero del record successivo che viene letto o scritto, quando si utilizzano le istruzione GET o PUT. Si noti che il primo record in un file è il numero 1.

Se il file è in formato binario o sequenziale, la funzione SEEK restituisce il numero di byte in cui si verificherà la successiva operazione di I/O. Si noti che il primo byte in un file è il numero 1.

Commenti

Il valore massimo che SEEK può restituire è 2147483647, dato che questa è la dimensione massima dei file supportati da QBasic.

Avvertenze

La funzione SEEK restituisce il valore 0, quando viene utilizzata con un numero associato ai dispositivi SCRN:, KYBD:, CONS: o LPTn, mentre restituisce valori significativi soltanto quando viene utilizzata con i file.

Esempio

Il programma di seguito riportato utilizza la funzione SEEK per determinare quale record è in uso in un file ad accesso casuale.

```
TYPE Pianeti            'tipo di dati contenente le informazioni sul pianeta
    nome AS STRING * 20                 'nome del pianeta
    diametro AS LONG                    'diametro del pianeta
    perielio AS DOUBLE                  'punto dell'orbita più vicino al sole
    afelio AS DOUBLE                    'punto dell'orbita più lontano dal sole
    maxTemp AS SINGLE                   'temperatura massima della superficie
    minTemp AS SINGLE                   'temperatura minima della superficie
    annoScoperta AS INTEGER             'anno di scoperta del pianeta
END TYPE

DIM pianeta AS Pianeti                  'definisce la variabile pianeta

CLS
OPEN "SOL.DAT" FOR RANDOM AS #1 LEN = 50    'apre il file di dati
```

```
FOR numPianeta% = 1 TO 9
    PRINT : PRINT : PRINT
    SEEK #1, numPianeta%                      'va al record corretto
    GET #1, , pianeta                         'legge il record
    PRINT "Numero del pianeta"; SEEK(1) - 1   'visualizza il numero del record

    PRINT : PRINT "Informazioni sul pianeta "; pianeta.nome
    PRINT STRING$(79, 45): PRINT

    'se il record è vuoto le informazioni saranno richieste all'utente
    IF pianeta.nome = STRING$(20, 0) THEN
        INPUT "Nome"; pianeta.nome
        INPUT "Diametro (km)"; pianeta.diametro
        INPUT "Perielio (km)"; pianeta.perielio
        INPUT "Afelio (km)"; pianeta.afelio
        INPUT "Temperatura massima (Celsius)"; pianeta.maxTemp
        INPUT "Temperatura minima (Celsius)"; pianeta.minTemp
        INPUT "Scoperto nel"; pianeta.annoScoperta
        PUT #1, numPianeta%, pianeta          'scrive le informazioni sul file
        PRINT : PRINT "Informazioni registrate": PRINT : PRINT
    ELSE                                      'visualizza le informazioni
        PRINT "Il diametro è"; pianeta.diametro; "chilometri"
        PRINT "Il perielio è"; pianeta.perielio; "chilometri"
        PRINT "L'afelio è"; pianeta.afelio; "chilometri"
        PRINT "La temperatura massima è"; pianeta.maxTemp; "gradi"
        PRINT "La temperatura minima è"; pianeta.minTemp; "gradi"
        PRINT "Scoperto nel"; pianeta.annoScoperta
    END IF
NEXT

CLOSE
```

SEEK (Istruzione)

Si consulti anche

GET (I/O su file), OPEN, PUT (I/O su file), SEEK (Funzione)

Scopo

L'istruzione SEEK imposta la posizione attuale in un file aperto, ovvero la posizione in corrispondenza della quale si verifica la successiva operazione di I/O (ad esempio con GET o PUT).

Sintassi

```
SEEK [#]numfile, posizione
```

numfile è il numero di un file aperto.

posizione è il numero del record o del byte sul quale ci si desidera posizionare all'interno del file e deve essere compreso nell'intervallo da 1 a 2147483647.

Utilizzo

```
OPEN "FILE.DAT" FOR RANDOM AS #1
SEEK 1, 128
```

Imposta la posizione attuale in corrispondenza del numero di record 128 per il file aperto come numero 1.

Descrizione

Le istruzioni di I/O, GET e PUT, consentono, se si desidera, di non specificare il record e il byte da cui leggere o in cui scrivere. In questo caso l'istruzione legge o scrive fino alla posizione attuale nel file, che viene aggiornata fino alla successiva operazione di lettura e scrittura. Questa posizione può essere utilizzata per accedere ai file in modo sequenziale, se ciò fosse necessario; tuttavia, l'istruzione SEEK consente di modificare la posizione attuale nel file, definendo un record o un byte qualsiasi. Dopo aver eseguito l'istruzione SEEK, la successiva operazione di I/O viene effettuata in corrispondenza della nuova posizione, a patto che non comprenda un record o un numero di byte specifici.

Per quanto riguarda i file ad accesso casuale, la posizione indicata, che diventa quella attuale, corrisponde al numero del record; il primo record in un file è il numero 1. Nel caso invece dei file in formato sequenziale o binario, la posizione specificata, che diventa quella attuale, è il numero del byte; il primo byte in un file è il numero 1.

Commenti

QBasic ignora l'istruzione SEEK, se il parametro numfile corrisponde ai dispositivi SCRN:, KYBD:, CONS:, COMn o LPTN:.

Errori

Se viene specificata una posizione inferiore o uguale a 0, QBasic visualizza il messaggio di errore: "Numero del record errato".

Esempio

Il programma di seguito riportato utilizza l'istruzione SEEK per modificare la posizione attuale, indicando un record specifico in un file ad accesso casuale.

```
TYPE Pianeti               'tipo di dati contenente le informazioni sul pianeta
    nome AS STRING * 20              'nome del pianeta
    diametro AS LONG                 'diametro del pianeta
    perielio AS DOUBLE               'punto dell'orbita più vicino al sole
    afelio AS DOUBLE                 'punto dell'orbita più lontano dal sole
    maxTemp AS SINGLE                'temperatura massima della superficie
    minTemp AS SINGLE                'temperatura minima della superficie
    annoScoperta AS INTEGER          'anno di scoperta del pianeta
END TYPE

DIM pianeta AS Pianeti                    'definisce la variabile pianeta

CLS
OPEN "SOL.DAT" FOR RANDOM AS #1 LEN = 50        'apre il file di dati

DO
    PRINT : PRINT : PRINT
    INPUT "Su quale pianeta si desidera ottenere le informazioni";
    numPianeta%  'ottiene il numero del record
  'esce se l'utente preme Invio o numPianeta% non rientra nei limiti corretti
    IF numPianeta% < 1 OR numPineta% > 9 THEN EXIT DO

    SEEK #1, numPianeta%                        'va al record corretto
    GET #1, , pianeta                           'legge il record

    PRINT : PRINT "Informazioni sul pianeta "; pianeta.nome
```

```
        PRINT STRING$(79, 45): PRINT

        'se il record è vuoto le informazioni saranno richieste all'utente
        IF pianeta.nome = STRING$(20, 0) THEN
            INPUT "Nome"; pianeta.nome
            INPUT "Diametro (km)"; pianeta.diametro
            INPUT "Perielio (km)"; pianeta.perielio
            INPUT "Afelio (km)"; pianeta.afelio
            INPUT "Temperatura massima (Celsius)"; pianeta.maxTemp
            INPUT "Temperatura minima (Celsius)"; pianeta.minTemp
            INPUT "Scoperto nel"; pianeta.annoScoperta
            PUT #1, numPianeta%, pianeta          'scrive le informazioni sul file
            PRINT : PRINT "Informazioni registrate": PRINT : PRINT
        ELSE                                      'visualizza le informazioni
            PRINT "Il diametro è"; pianeta.diametro; "chilometri"
            PRINT "Il perielio è"; pianeta.perielio; "chilometri"
            PRINT "L'afelio è"; pianeta.afelio; "chilometri"
            PRINT "La temperatura massima è"; pianeta.maxTemp; "gradi"
            PRINT "La temperatura minima è"; pianeta.minTemp; "gradi"
            PRINT "Scoperto nel"; pianeta.annoScoperta
        END IF
    LOOP
    CLOSE
```

UNLOCK

Si consulti anche

LOCK, OPEN

Scopo

L'istruzione UNLOCK consente ad altri programmi in rete di accedere a un file condiviso o a una sua parte; può essere utilizzata per consentire l'accesso a un record che era stato precedentemente bloccato durante l'esecuzione di un'operazione di I/O critica.

Sintassi

```
UNLOCK [#]numfile [,{record | [inizio] TO fine}]
```

numfile è il numero di un file aperto.
record è il numero del record o del byte da sbloccare.
inizio e fine sono interi che indicano l'intervallo dei record o dei byte da sbloccare.

Utilizzo

```
UNLOCK #1
```

Sblocca tutto il file aperto come numero 1.

```
OPEN "VENDITE.DAT" FOR RANDOM AS #4
.
.
.
UNLOCK #4, 10
```

Sblocca il record 10 del file ad accesso casuale aperto come numero 4.

Descrizione

UNLOCK può essere utilizzata in un ambiente di rete in cui diversi utenti o operazioni condividono gli stessi file. Dopo aver eseguito l'istruzione LOCK per impedire il sorgere di conflitti quando diversi programmi cercano di accedere contemporaneamente agli stessi dati, è necessario utilizzare l'istruzione UNLOCK per sbloccare il file.

In un file ad accesso casuale è sufficiente specificare i record da sbloccare; il primo record nel file è quello numero 1. In un file binario, è necessario specificare i byte da sbloccare; il primo byte nel file è quello numero 1. Nei file ad accesso casuale o binari è possibile sbloccare singoli record o byte, mentre nei file sequenziali può essere sbloccato soltanto tutto il file.

Commenti

Le istruzioni LOCK e UNLOCK devono essere perfettamente congruenti, ovvero non è possibile combinare le istruzioni UNLOCK corrispondente a più di un'istruzione LOCK per avere un'unica istruzione UNLOCK, ma è necessario invece utilizzare quest'ultima in modo che corrisponda esattamente all'originale istruzione LOCK. Ad esempio, le due istruzioni:

```
LOCK #3, 1 TO 10
LOCK #3, 11 TO 20
```

richiedono due istruzioni UNLOCK:

```
UNLOCK #3, 1 TO 10
UNLOCK #3, 11 TO 20
```

non è quindi sufficiente combinare semplicemente le due istruzioni UNLOCK in una singola istruzione, come:

```
UNLOCK #3, 1 TO 20
```

Si noti che, anche se QBasic consente di immettere l'istruzione sopra riportata, senza restituire alcun messaggio di errore, essa non ha alcun effetto e i 20 byte del record rimangono bloccati.

Errori

Se l'istruzione UNLOCK viene specificata prima di utilizzare LOCK, QBasic visualizza il messaggio di errore: "Permesso negato".

Avvertenze

Microsoft supporta QBasic soltanto sulle reti Microsoft LAN Manager, IBM-PC Network, MS-Network e sistemi compatibili. Sebbene siano stati ottenuti dei risultati anche con altre reti, quali Novell e 3Com, possono in ogni caso sorgere dei problemi.

Se dopo che un file viene bloccato si verifica un calo di tensione o si esegue il reset del computer prima che venga eseguita la corrispondente istruzione UNLOCK, si potrebbero perdere i dati presenti nel file bloccato. Per ridurre al minimo la possibilità di perdere dati, è necessario utilizzare LOCK per bloccare i record solo fino a quando è strettamente necessario.

Esempio

Il programma di seguito riportato utilizza l'istruzione LOCK per impedire che un altro programma acceda al file CONFIG.DAT durante l'esecuzione. Al termine del programma il file viene sbloccato immettendo la corrispondente istruzione UNLOCK.

```
CLS
TYPE TipoConfig
     normale AS INTEGER              'colore dell'area interna del riquadro
     altaInt AS INTEGER               'colore della lettera in alta intensità
     invAltaInt AS INTEGER      'colore dell'inverso della lettera in alta
                                'intensità
     inverso AS INTEGER               'colore della barra in alta intensità
     aiuto AS INTEGER                 'colore della riga di aiuto
     cursoreNorm AS INTEGER     'dimensione del cursore in modo sovrascrittura
     cursoreIns AS INTEGER      'dimensione del cursore in modo Inserimento
END TYPE

DIM config AS TipoConfig                  'definisce la variabile record config

CLS
PRINT "Immettere i nuovi colori."
PRINT

OPEN "CONFIG.DAT" FOR BINARY ACCESS READ WRITE LOCK WRITE AS #1 'apre il file
LOCK #1                    'limita l'accesso all'intero file
GET #1, 1, config          'legge le informazioni sulla configurazione corrente


PRINT "Colore normale = "; config.normale;
INPUT " Cambia in: ", config.normale

PRINT "Colore alta intensità = "; config.altaInt;
INPUT " Cambia in: ", config.altaInt

PRINT "Inverso colore alta intensità = "; config.invAltaInt;
INPUT " Cambia in: ", config.invAltaInt

PRINT "Inverso colore = "; config.inverso;
INPUT " Cambia in: ", config.inverso

PRINT "Colore aiuto = "; config.aiuto;
INPUT " Cambia in: ", config.aiuto

PRINT "Dimensione del cursore in modalità sovrascrittura = ";
config.cursoreNorm;
INPUT " Cambia in: ", config.cursoreNorm

PRINT "Dimensione del cursore in modalità inserimento = "; config.cursoreIns;
INPUT " Cambia in: ", config.cursoreIns

PUT #1, 1, config                   'scrive le informazioni nel file
UNLOCK #1                          'rimuove i limiti di accesso all'intero file

CLOSE #1

END
```

WIDTH (I/O su file)

Si consulti anche

LPOS, LPRINT, POS, PRINT, TAB

Scopo

L'istruzione WIDTH imposta la larghezza massima della riga per i file sequenziali; può essere ad esempio utilizzata per limitare la lunghezza delle righe in un file contenente un prospetto.

Sintassi

```
WIDTH [#numfile | unità], larghezza
```

numfile è il numero di un file sequenziale aperto.
unità è un'espressione stringa che indica il nome di un dispositivo di QBasic.
larghezza è il valore della larghezza desiderata (in caratteri) di una riga relativa sia a numfile che a unità.

Utilizzo

```
WIDTH #1, 40
```

Imposta a 40 caratteri la larghezza del file specificato.

```
WIDTH "CONS:", 32
```

Imposta a 32 caratteri la larghezza del dispositivo CONS:.

Descrizione

Generalmente, i dati vengono scritti in un file sequenziale senza assegnare alcuna formattazione. Tuttavia, è possibile utilizzare l'istruzione WIDTH per impedire che il testo venga scritto oltre il margine destro specificato; questa istruzione consente di realizzare questa operazione, portando qualsiasi riga troppo lunga sulla riga successiva. Nella Figura 17.11 viene illustrato l'effetto provocato dall'istruzione WIDTH.

L'utilizzo di questa istruzione con un nome di file consente di impostare la larghezza delle righe di un file aperto o di un dispositivo di QBasic, aperto come file. Questo rappresenta in effetti l'unico metodo per modificare la larghezza delle righe di un file. Il file o il dispositivo devono essere aperti, prima di poter immettere l'istruzione WIDTH.

L'utilizzo di questa istruzione con un nome di dispositivo consente di impostare la larghezza delle righe per il dispositivo stesso di QBasic, ma non per quelli DOS. L'istruzione WIDTH non alcun effetto sull'output del dispositivo, fino a quando esso non viene aperto la volta successiva.

Errori

Se il dispositivo specificato non rientra tra quelli previsti da QBasic (SCRN:, KYBD:, CONS:, COMn o LPTN:), il programma visualizza il messaggio di errore: "Chiamata di funzione non lecita".

Esempio

Il programma di seguito riportato crea un prospetto, le cui righe non superano gli 80 caratteri.

```
WIDTH, #1, 25
esempio$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGH"
PRINT #1, esempio$
```

A	B	C	D	E	F	G	H	I	L	M	N	O	J	K	P	Q	R	S	T	U	V	W	X	Y
Z	A	B	C	D	E	F	G	H	I	L	M	N	O	J	K	P	Q	R	S	T	U	V	W	X
Y	Z	A	B	C	D	E	F	G	H															

Figura 17.11 L'utilizzo dell'istruzione WIDTH

```
OPEN "VENDITE.RPT" FOR OUTPUT AS #1  'apre il file sulle vendite
WIDTH #1, 80                         'imposta la larghezza della linea a 80

PRINT #1, "Rapporto sulle vendite del "; DATE$  'stampa l'intestazione

FOR regione% = 1 TO 21
    PRINT #1, USING "Regione ##: "; regione%   'stampa l'intestazione di riga
    ImportoVendite% = RND * 50                 'calcola vendite  fittizie
    FOR i% = 1 TO ImportoVendite%
        PRINT #1, "* ";                        'scrive il grafico nel file
    NEXT
    PRINT #1,                          'va alla riga successiva del rapporto
NEXT                                           'va alla regione successiva

CLOSE
```

WRITE

Si consulti anche

INPUT #, PRINT #

Scopo

L'istruzione WRITE # scrive dei dati delimitati e non formattati in un file sequenziale; può essere utilizzata per scrivere velocemente i dati in un file, in modo che possano essere facilmente letti da un'istruzione INPUT #.

Sintassi

```
WRITE #numfile[,espressione[,espressione...]]
```

numfile è il numero del file aperto.

espressione è un'espressione numerica o stringa che contiene il valore da scrivere nel file; le voci multiple devono essere separate da virgole.

Utilizzo

```
WRITE #2, NomeDegente$, DataNascitaDegente%, ImportoDovuto#
```

Scrive i valori delle tre variabili nel file aperto come numero 2; il valore di NomeDegente$ viene racchiuso tra doppi apici, mentre tutti i valori sono separati da virgole.

Descrizione

L'istruzione INPUT # richiede che i dati siano formattati e delimitati. Se viene utilizzata l'istruzione PRINT # per scrivere i dati in un file, è necessario includere esplicitamente spazi, doppi apici o virgole come delimitatori. Tutto ciò può risultare complicato, come si può notare nel seguente esempio:

```
PRINT #2, CHR$(34); NomeDegente$; CHR$(34); ","; DataNascitaDegente%; ",";
ImportoDovuto#
```

La stessa operazione può essere eseguita utilizzando un'istruzione WRITE # più semplice e leggibile, come nell'esempio di seguito riportato:

```
WRITE #2, NomeDegente$, DataNascitaDegente%, ImportoDovuto#
```

Il numero di file specificato è il numero associato al file in cui vengono scritti i dati; esso deve essere stato aperto come file sequenziale, utilizzando la parola chiave OUTPUT o quella APPEND in un'istruzione OPEN.

Il valore scritto nel file può essere specificato da una costante (testo tra doppi apici, un valore numerico o una costante definita da CONST) o da un'espressione di qualsiasi tipo semplice, quale un elemento di array, un elemento di un record o una variabile semplice, oppure un'espressione stringa o matematica che utilizza le funzioni di QBasic. Se non viene utilizzato alcun parametro, l'istruzione scrive una riga nel file.

Quando WRITE # scrive i dati in un file, utilizza la minima quantità di spazio, consentendo tuttavia all'istruzione INPUT # di leggere successivamente ciò che era stato scritto. Come riportato nella Figura 17.12, WRITE # racchiude le variabili stringa tra doppi apici, ma al contrario di ciò che si verifica con PRINT # le variabili numeriche vengono scritte in un file senza spazi iniziali o finali. Le variabili numeriche negative vengono precedute dal segno meno e tutte le variabili vengono separate da virgole.

Dopo aver scritto nel file l'ultimo valore specificato, l'istruzione WRITE # inserisce un carattere di avanzamento riga. Quindi, le istruzioni WRITE # e PRINT # successive scrivono i dati in corrispondenza di questa nuova riga. Al contrario dell'istruzione PRINT #, WRITE # inserisce sempre un carattere di avanzamento riga.

Commenti

WRITE # non dispone di una corrispondente istruzione WRITE # USING che fornisca l'output formattato; a questo scopo, è necessario utilizzare PRINT # USING.

Dato che l'istruzione WRITE # racchiude le stringhe tra doppi apici, INPUT # non è in grado di leggere correttamente le stringhe che sono state scritte nel file utilizzando WRITE # e che hanno già i doppi apici. In questo caso, è necessario utilizzare l'istruzione LINE INPUT #.

```
nomePaziente$ = "Rossi, Giorgio"
compleanno! = 12170
ultimoImportoDovuto# = 150000
```

"	R	o	s	s	i	,		G	i	o	r	g	i	o	"	,	1	2	1	7	0	,	1	5	0	0	0	0

Figura 17.12 L'utilizzo di WRITE # per scrivere i dati in un file

Esempio

Il programma di seguito riportato utilizza l'istruzione WRITE per scrivere il contenuto della schermata visualizzata in un file; ogni riga è racchiusa tra doppi apici, in modo che l'istruzione INPUT # possa successivamente leggere il file, una riga alla volta.

```
OPEN "SCHERMO.DAT" FOR APPEND AS #1

FOR y = 1 TO 25
    lineeSchermo$ = ""

    FOR x = 1 TO 80
        lineeSchermo$ = lineeSchermo$ + CHR$(SCREEN(y, x))
    NEXT

    WRITE #1, lineeSchermo$
NEXT

WRITE #1, STRING$(80, 196)
WRITE #1, "Questa schermata è stata realizzata il ", DATE$, " alle ", TIME$,
"."
WRITE #1, STRING$(80, 196)

CLOSE
```

Parte quarta

Sviluppo

Capitolo 18

Il DOS e la gestione dei programmi

INTRODUZIONE

QBasic presenta molti comandi e funzioni che consentono di controllare l'interazione tra il proprio programma e l'ambiente DOS. Ad esempio le routine di gestione dei programmi illustrate in questo capitolo permettono di eseguire le seguenti operazioni:

- utilizzare i diversi metodi che consentono di uscire da un programma QBasic;
- passare il controllo a un altro programma QBasic;
- esaminare la copia di QBasic dell'ambiente DOS, modificandolo o aggiungendo delle variabili;
- creare, modificare o eliminare una directory su disco;
- cancellare i file da un disco;
- eseguire il processore dei comandi del DOS, eseguire qualsiasi comando del DOS o altri programmi e quindi riprendere l'esecuzione di un programma QBasic;
- inviare istruzioni di controllo oppure riceverle dal driver del dispositivo.

Nella Tabella 18.1 sono riportate in ordine alfabetico le istruzioni e le funzioni relative alla gestione dei programmi QBasic.

IL SEPARATORE DI ISTRUZIONI E I COMMENTI

Sulla stessa riga di programma è possibile immettere diverse istruzioni di QBasic, separandole con i due punti (:). Generalmente ciò rende il programma di più difficile

Tabella 18.1 Le istruzioni e le funzioni di QBasic relative alla gestione dei programmi

Istruzione o funzione	Descrizione
CHDIR	Cambia la directory attuale predefinita dell'unità disco specificata.
CLEAR	Reinizializza tutte le variabili del programma, chiude i file e imposta le dimensioni dello stack di QBasic.
: (due punti)	Separa le istruzioni multiple sulla stessa riga di programma.
END	Termina un programma e chiude tutti i file aperti.
ENVIRON	Aggiunge, elimina o modifica una variabile di ambiente.
ENVIRON$	Restituisce il valore della copia di QBasic di una variabile di ambiente del DOS.
IOCTL	Invia una stringa di controllo al driver di dispositivo.
IOCTL$	Riceve una stringa di controllo dal driver di dispositivo.
KILL	Cancella un file su disco.
MKDIR	Crea la directory specificata su disco.
REM	Inizia una riga di commento. Anche un apice può iniziare una riga di commento o un commento alla fine di una riga eseguibile.
RMDIR	Elimina una directory da disco. La directory non deve contenere alcun file.
RUN	Avvia nuovamente il programma attuale (facoltativamente, da un numero di riga specificato), oppure esegue un altro programma; tuttavia il secondo programma non può accedere ai dati dal primo.
SHELL	Esegue una copia del COMMAND.COM, il processore di comandi del DOS, con cui è possibile eseguire qualsiasi comando o programma del DOS (memoria permettendo).
STOP	Termina l'esecuzione di un programma QBasic, ma non chiude i file aperti.
SYSTEM	Termina l'esecuzione di un programma QBasic e chiude tutti i file aperti.

lettura, ma talvolta non è così, ad esempio quando vengono utilizzate diverse istruzioni PRINT per inserire alcune righe vuote:

```
PRINT : PRINT : REM saltare due righe
```

Questa riga comprende due istruzioni PRINT e un'istruzione REM (osservazione o commento), che illustra lo scopo della riga. Per specificare un commento è anche possibile utilizzare un apice, come nella seguente istruzione:

```
PRINT : PRINT 'saltare due righe
```

Si noti che non è necessario inserire i due punti prima del commento che presenta l'apice, su una riga che contiene le istruzioni eseguibili. Un commento può occupare da solo tutta una riga, ma se segue un codice eseguibile sulla stessa riga, deve essere preceduto da un apice o da due punti e dalla parola chiave REM.

USCITA DA UN PROGRAMMA QBASIC

QBasic richiama implicitamente l'istruzione END, dopo l'ultima istruzione di un programma. END chiude tutti i file e termina l'esecuzione. Questa istruzione può essere inserita in qualsiasi parte del programma, come riportato nel seguente esempio:

```
DO
   VisualizzaMenu
   INPUT risp$
   IF UCASE$(risp$) = "X" THEN END
   IF UCASE$(risp$) = "A" THEN AggiungiFile
LOOP
```

Questo ciclo esegue END soltanto se l'utente immette x o X. La parola chiave END viene anche utilizzata con le istruzioni DEF FN, FUNCTION, IF, SELECT CASE, SUB e TYPE...END TYPE per indicare la fine di queste strutture o definizioni.

UTILIZZO DEI COMANDI DOS IN UN PROGRAMMA

I comandi CHAIN e RUN possono essere entrambi utilizzati per eseguire un secondo programma da un programma QBasic. CHAIN consente al secondo programma di accedere a qualsiasi variabile dichiarata in un'istruzione COMMON, RUN invece può avviare un secondo programma o riprendere l'esecuzione del programma in uso, dall'inizio oppure partendo dal numero di riga specificato. CHAIN e RUN possono eseguire soltanto i file sorgente Basic.

Se si desidera che il programma esegua un comando DOS o un file batch, è possibile specificarlo in una chiamata all'istruzione SHELL. Questa istruzione consente di uscire dal programma, eseguire il COMMAND.COM del DOS, effettuare l'attività specificata e quindi di ritornare al programma. Se nella chiamata a SHELL non viene specificato un comando o un file batch, questa istruzione esce dal programma ed esegue il processore dei comandi, fino a quando l'utente non immette il comando EXIT.

REINIZALIZZAZIONE DELLE VARIABILI E IMPOSTAZIONE DELLE DIMENSIONI DELLO STACK

L'istruzione CLEAR imposta tutte le variabili numeriche e gli elementi numerici degli array a 0, assegna una stringa nulla ("") a tutte le variabili stringa e agli elementi stringa degli array e chiude tutti i file. Se viene eseguita prima di RUN, è possibile avviare nuovamente il programma ripulendo lo stack.

CLEAR consente inoltre di ripulire lo stack e di impostarne le dimensioni; lo stack è l'area in memoria utilizzata da QBasic per memorizzare il contenuto delle variabili presenti nelle chiamate alle subroutine e alle funzioni. Ad esempio, l'istruzione:

```
CLEAR , , 8000
```

reinizializza tutte le variabili e imposta lo stack a 8000 byte. Potrebbe essere necessario aumentare le dimensioni dello stack, se il programma contiene numerose subroutine o procedure nidificate oppure se vengono utilizzate procedure *ricorsive* (sottoprogrammi o funzioni che richiamano se stesse). CLEAR non può essere utilizzata all'interno di una procedura o di una subroutine: lo stack mantiene la posizione in corrispondenza della quale deve ritornare la procedura o la subroutine, e quindi dopo che CLEAR lo ha cancellato, non è più possibile restituire il controllo all'istruzione chiamante.

LETTURA E IMPOSTAZIONE DELLE VARIABILI DI AMBIENTE

I programmi QBasic possono ottenere e modificare le informazioni relative all'ambiente del DOS, ovvero quell'area di memoria che il DOS copia, durante l'esecuzione, nell'area dati di un programma QBasic. La copia consiste di una serie di variabili associate a stringhe di informazioni.

QBasic presenta un'istruzione e una funzione che interagiscono con la copia di programma dell'ambiente. L'istruzione ENVIRON sostituisce le informazioni in una variabile dell'ambiente DOS; ad esempio:

```
ENVIRON "PATH=C:\IMP;C:\TEMP"
```

assegna alla variabile PATH il valore C:\AIMPT;C:\TEMP.

La funzione ENVIRON$ in QBasic restituisce il valore di una variabile di ambiente; mentre l'istruzione ENVIRON$(1) restituisce la prima variabile e il relativo valore, ENVIRON$(2) restituisce il secondo e così via. Quindi, se PATH è la terza variabile nell'ambiente, l'istruzione:

```
PRINT ENVIRON$(3)
```

visualizza PATH=C:\IMP;C:\TEMP

È possibile utilizzare un nome di variabile con l'istruzione ENVIRON$ per ottenere il valore della variabile specificata. L'istruzione di seguito riportata visualizza la stessa stringa dell'esempio precedente.

```
PRINT ENVIRON$("PATH")
```

È necessario tenere presente che il programma QBasic riceve soltanto una copia dell'ambiente originale del DOS; qualsiasi modifica specificata nelle istruzioni ENVIRON non ha effetto sull'ambiente originale del DOS, che viene ripristinato quando il programma termina l'esecuzione. Tuttavia, se il proprio programma esegue un altro programma con il comando CHAIN o RUN oppure con l'istruzione SHELL, il nuovo programma riceve una copia dell'ambiente in uso al proprio programma, compresa qualsiasi modifica o aggiunta apportata.

COMUNICAZIONE CON I DRIVER DEI DISPOSITIVI

L'istruzione IOCTL ("controllo I/O") invia una stringa di istruzioni ai driver dei dispositivi, che sono programmi speciali che il DOS installa in memoria (generalmente

mediante l'istruzione DEVICE nel file CONFIG.SYS). Il programma interpreta le istruzioni relative a un dispositivo (ad esempio la stampante, la porta di comunicazione o il mouse) e le traduce nelle istruzioni di basso livello utilizzate dall'hardware. Successivamente, è possibile richiedere che il driver di dispositivo riferisca lo stato dello stesso (ad esempio, se la stampante è pronta o meno ad accettare una parte ulteriore di testo). Quando viene specificato il numero di file associato al dispositivo aperto, l'istruzione IOCTL$ restituisce il prospetto di stato corrispondente, se il dispositivo supporta le istruzione IOCTL. Purtroppo, la maggior parte dei dispositivi (compresi LPT1, LPT2, COM1, COM2 e così via) non sono progettati per accettare queste istruzioni.

GESTIONE DEI FILE E DELLE DIRECTORY

QBasic presenta diverse istruzioni che consentono di gestire i file e tre istruzioni che consentono invece di gestire le directory: MKDIR crea una directory; CHDIR modifica la directory in uso; RMDIR rimuove una directory. Queste istruzioni sono simili a quelle corrispondenti del DOS, anche se non possono essere abbreviate allo stesso modo.

Bibliografia

Lesser, Murray. *Advanced QuickBASIC 4.0: Language Extension with Modular Tools.* New York, N.Y.: Bantam Computer Books, 1988

Prata, Stephen, with Harry Henderson. *The Waite Group's Microsoft QuickBASIC Primer Plus.* Redmond, Wash.: Microsoft Press, 1990

CHDIR

Si consulti anche

MKDIR, RMDIR, SHELL, SYSTEM

Scopo

L'istruzione CHDIR cambia la directory in uso; può essere utilizzata per cambiare la directory di default per i comandi relativi ai file (quali OPEN o CHAIN), in modo che il programma non debba specificare un percorso completo della directory.

Sintassi

```
CHDIR directory
```

directory è un'espressione stringa con un massimo di 63 caratteri che indica una nuova directory o un nuovo percorso.

Utilizzo

```
TEMPDIR$ = "C:\TEMP"
CHDIR tempdir$
```

Modifica la directory attuale sull'unità disco C nella directory specificata nella variabile stringa tempdir$.

Descrizione

Il DOS mantiene una directory di default per ogni unità disco nel sistema, che viene utilizzata dal DOS e da QBasic, se non vengono specificate le informazioni relative a un percorso. Ad esempio, se la directory in uso sull'unità C (quella corrente) è TEMP, l'istruzione FILES elenca il contenuto della directory C:\TEMP.

Il DOS tiene traccia sia dell'unità disco attuale che della directory in uso su ogni unità; se quest'ultima viene modificata, ciò non significa che venga cambiata anche l'unità. Ad esempio, se l'unità disco attiva è A, le istruzioni:

```
CHDIR "C:\TEMP"
FILES
```

modificano la directory attuale sull'unità C, ma elencano la directory corrente dell'unità A, non quella C:\TEMP. L'istruzione CHDIR non consente di modificare l'unità disco attiva, a questo scopo è necessario sfruttare l'istruzione SHELL.

I programmi creati non devono presupporre una determinata directory o unità, ma devono invece impostare esplicitamente la directory e l'unità in uso.

Commenti

L'istruzione CHDIR opera essenzialmente come il comando CHDIR del DOS, l'unica differenza è che QBasic non riconosce l'abbreviazione CD.

Errori

Se nell'istruzione CHDIR viene specificato un percorso che non esiste, QBasic visualizza il messaggio di errore: "Percorso di ricerca non trovato". Se viene indicata un'espressione non stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi". Se invece non viene specificato un parametro, QBasic visualizza il messaggio di errore: "Era atteso: espressione".

Esempio

Il programma di seguito riportato richiede all'utente di immettere il percorso di una directory e quindi utilizza l'istruzione CHDIR per renderla attiva. Poi, l'istruzione FILES visualizza il contenuto della directory indicata.

```
INPUT "Immettere la directory desiderata: ", dir$
CHDIR dir$                                    'cambia la directory attuale

'visualizza il contenuto della directory
PRINT "*** Elenco dei file in "; dir$; " ***": PRINT
FILES
```

CLEAR

Si consulti anche

DECLARE, DIM, ERASE, FRE, REDIM

Scopo

L'istruzione CLEAR reinizializza tutte le variabili di programma (impostando i valori numerici a 0 e le stringhe alla stringa nulla "") e lo stack, chiude tutti i file e modifica, se richiesto, le dimensioni dello stack. CLEAR può essere utilizzata per aumentare le dimensioni dello stack, se il programma contiene procedure ricorsive o nidificate oppure numerosi parametri e variabili automatiche.

Sintassi

```
CLEAR [,, stack]
```

stack è il numero di byte (nell'intervallo da 0 a 32767) da allocare per lo stack utilizzato dal programma. La variabile **stack** deve sempre essere preceduta da due virgole.

Utilizzo

```
CLEAR
```

Reinizializza tutte le variabili, ripulisce lo stack e chiude tutti i file aperti.

```
CLEAR ,, 1024
```

Reinizializza tutte le variabili, ripulisce lo stack, chiude tutti i file aperti e imposta le dimensioni dello stack a 1024 byte.

Descrizione

L'utilizzo principale dell'istruzione CLEAR è quello di reinizializzare tutte le variabili (compresi gli elementi degli array) a 0 (per i valori numerici) o alla stringa nulla (per i valori stringa). In un programma che rileva un errore molto grave (ad esempio che comporta l'alterazione dei dati), è possibile utilizzare CLEAR per eliminare tutti i valori delle variabili potenzialmente errate, riavviando quindi il programma dall'inizio e ripulendo lo stack (la stessa operazione può essere eseguita in un gioco, per avviarne uno nuovo su richiesta). CLEAR chiude tutti i file.

Questa istruzione non deve essere utilizzata semplicemente per reinizializzare gli array e le altre variabili in un momento successivo, dato che esegue questa operazione con tutte le variabili, non soltanto con quelle che si desidera utilizzare nuovamente. A questo scopo è invece possibile sfruttare ERASE, che consente di reinizializzare gli array specificati senza per questo influenzare le altre variabili.

CLEAR può anche essere sfruttata per modificare le dimensioni dello stack utilizzato dal programma. In questo caso, è necessario assicurarsi di richiamare l'istruzione all'inizio del programma, prima di dimensionare gli array o di inizializzare le altre variabili, dato che CLEAR le reinizializza tutte.

QBasic addiziona la quantità necessaria di spazio dello stack e quella specificata per calcolare le dimensioni di tutto lo stack; il valore di default è di 2048 byte. Le dimensioni rimangono impostate all'ultimo valore specificato fino a quando non viene eseguito un programma diverso.

Errori

Se CLEAR viene utilizzata in una subroutine, QBasic visualizza il messaggio di errore: "RETURN senza GOSUB". Se invece questa istruzione viene inserita in una procedura, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita".

Esempio

Il programma di seguito riportato illustra l'utilizzo dell'istruzione CLEAR all'inizio di un programma per impostare le dimensioni dello stack. Viene utilizzata la funzione ricorsiva Fattoriale# per calcolare il fattoriale di un numero (il prodotto di tutti i valori da 1 fino al numero). Il programma richiama la funzione FRE con il parametro -2, in modo che restituisca la quantità di spazio attualmente disponibile nello stack. Si noti che lo spazio dello stack diminuisce con ogni chiamata ricorsiva alla funzione Fattoriale#.

```
DECLARE FUNCTION fattoriale# (num%)

INPUT "Cambia le dimensioni per lo stack (s/n) "; risp$
IF UCASE$(risp$) = "S" THEN
   INPUT "Dimensioni dello stack: ", dimensioni
   CLEAR , , dimensioni
END IF

PRINT "Spazio disponibile per lo stack "; FRE(-2); " byte."

INPUT "Il fattoriale di quale numero "; numero%
fatt# = fattoriale#(numero%)
PRINT "Il fattoriale di "; numero%; " è "; fatt#


FUNCTION fattoriale# (num%)
    PRINT "Spazio rimasto sullo stack: "; FRE(-2)
    IF num% = 0 THEN
       fattoriale# = 1
    ELSE fattoriale# = (num% * fattoriale#(num% - 1))
    END IF
END FUNCTION
```

: (DUE PUNTI)

Si consulti anche

DATA, REM

Scopo

I due punti (:) separano le istruzioni di QBasic su una riga del codice sorgente.

Sintassi

```
istruzione : istruzione [: istruzione...]
```

istruzione è una qualsiasi istruzione valida di QBasic.

Utilizzo

```
FOR cliente% = 1 TO numClienti%
   PRINT InfoClt(cliente%).NomeClt: PRINT InfoClt(cliente%).indirizzo
   PRINT InfoClt(cliente%).città: PRINT InfoClt(cliente%).cop
NEXT cliente%
```

Utilizza i due punti per inserire diverse istruzioni PRINT su una riga.

Descrizione

Se sulla stessa riga viene inserita più di un'istruzione, è necessario separarle mediante i due punti. Questo tipo di sintassi rende spesso il codice più difficile da leggere e, soprattutto, rende poco chiara l'operazione di salto e le istruzioni di esecuzione del ciclo; proprio per queste ragioni è meglio utilizzare i due punti con parsimonia, sfruttando invece le istruzioni di blocco e i rientri per meglio mettere in luce la struttura di un ciclo, le relative condizioni e le istruzioni che ne vengono influenzate.

Commenti

QBasic non consente di immettere una riga di lunghezza superiore ai 256 caratteri, anche se accetta righe più lunghe caricate dai file su disco. Se alla fine di una riga creata all'esterno dell'editor di QBasic viene inserito un trattino di sottolineatura (_), QBasic unisce quella riga con quella seguente, indipendentemente dalla lunghezza della singola riga così realizzata. Si noti che non è possibile utilizzare il trattino di sottolineatura per continuare un'istruzione REM o DATA.

I due punti possono anche essere sfruttati in un programma per indicare che la parola precedente è un'etichetta di riga.

Esempio

Il programma di seguito riportato illustra l'utilizzo delle istruzioni multiple su una riga. In questo caso, la seconda istruzione PRINT, presente su alcune righe, assegna un'interlinea doppia al testo; l'inserimento di una seconda istruzione PRINT rende il listato più compatto, ma non illeggibile.

```
TYPE Informazione
     nomeCliente AS STRING * 20
     via AS STRING * 15
     citta AS STRING * 15
     cap AS INTEGER
END TYPE

numClienti% = 15
DIM infoCliente(numClienti%) AS Informazione

PRINT "Lista degli indirizzi dei clienti": PRINT        'doppio spazio
PRINT "=================================": PRINT
PRINT TAB(8); DATE$: PRINT
PRINT "=================================": PRINT

FOR clienti% = 1 TO numClienti%
     PRINT infoCliente(clienti%).nomeCliente: PRINT infoCliente(clienti%).via
     PRINT infoCliente(clienti%).citta: PRINT infoCliente(clienti%).cap
NEXT clienti%
```

END

Si consulti anche

FUNCTION, GOSUB...RETURN, GOTO, IF, SELECT CASE, SUB, TYPE...END TYPE

Scopo

L'istruzione END, oltre a concludere un programma QBasic, consente anche di terminare la definizione di un tipo, una funzione oppure la definizione di un sottoprogramma o di un'istruzione di blocco.

Sintassi

```
END
```

Utilizzo

```
PRINT "Questo è un programma molto breve."
END
```

Termina l'esecuzione del programma in corrispondenza della seconda riga.

Descrizione

L'istruzione END chiude tutti i file aperti e termina l'esecuzione di un programma QBasic.

Se il proprio programma comprende delle subroutine, è sufficiente inserire un'istruzione END tra il programma principale e la prima subroutine per impedire che quest'ultima venga eseguita inavvertitamente. QBasic esegue un'istruzione END implicita alla fine di ogni programma, per questa ragione non è necessario utilizzare END come ultima istruzione, ma è comunque opportuno sfruttarla migliorare la leggibilità del programma.

Esempio

Il programma di seguito riportato utilizza due istruzioni END: la prima all'interno del ciclo provoca l'uscita immediata del programma, se l'utente risponde alla richiesta digitando la lettera S.

```
CONST MIGLIAAKM = 1.61
PRINT "Questo programma converte miglia orarie in chilometri orari."
DO
   INPUT "Vuoi terminare "; risp$
   IF UCASE$(risp$) = "S" THEN END
   INPUT "A che velocità stai guidando (im miglia orarie) "; mph
   PRINT "Cioè "; mph * MIGLIAAKM; " chilometri orari."
LOOP UNTIL mph = 0
PRINT "Sei senza carburante !"
END
```

ENVIRON

Si consulti anche

ENVIRON$, SHELL

Scopo

L'istruzione ENVIRON modifica il valore della variabile d'ambiente oppure aggiunge una voce alla copia del programma dell'ambiente DOS (un elenco delle variabili a livello di sistema e dei relativi valori, che QBasic mette a disposizione per qualsiasi programma in fase di esecuzione); può essere utilizzata per modificare le informazioni (ad esempio il percorso di ricerca della directory) riguardanti un programma eseguito dal programma attivo mediante il comando RUN o i comandi DOS eseguiti da un'istruzione SHELL.

Sintassi

```
ENVIRON "espressionestringa[=] [testo |;]"
```

espressionestringa è una stringa che identifica la variabile di ambiente. Per assegnare un valore a questa variabile è sufficiente inserire dopo il nome un segno di uguale o uno spazio specificando quindi testo, che è il valore della variabile.

Se al posto di testo viene inserito un punto e virgola, oppure se non viene indicata alcuna parte di testo, ENVIRON cancella la variabile dall'ambiente.

Utilizzo

```
ENVIRON "PATH=C:\QBIMP;C:\QBIMP\TEST;C:\IMP\TEST"
```

Modifica il valore della variabile di ambiente PATH nella copia di QBasic dell'ambiente.

```
ENVIRON "SORGENTE="
```

Cancella la variabile di ambiente SORGENTE dalla copia di QBasic dell'ambiente operativo.

Descrizione

Il DOS gestisce un'area di memoria che contiene diverse variabili a cui può accedere il programma. Queste variabili comprendono il PATH, ovvero l'elenco delle directory che il DOS ricerca per i file eseguibili, COMSPEC, ovvero la posizione e il nome del processore di comando e PROMPT, ovvero la stringa di caratteri utilizzata dal DOS per le proprie richieste. Ogni nome di queste variabili viene associato alla stringa che ne contiene il valore. Per visualizzare un elenco delle variabili di ambiente attualmente definite, è sufficiente digitare SET al prompt del DOS.

Durante l'esecuzione di un programma QBasic, quest'ultimo riceve una copia dell'ambiente DOS, ovvero un elenco delle variabili e dei relativi valori. L'istruzione ENVIRON consente di modificare i valori delle variabili e di cancellarne o aggiungerne alcune; si tenga presente che viene modificata soltanto una copia dell'ambiente DOS, mentre quella originale rimane invariata. Tuttavia, le modifiche che il programma

apporta vengono passate a qualsiasi programma da esso eseguito e vengono ereditate dagli altri programmi eseguiti nell'ambiente di QBasic.

Commenti

Per default, l'area di memoria relativa all'ambiente è generalmente soltanto di 128 o 256 byte, quindi l'utilizzo frequente dell'istruzione ENVIRON provoca il messaggio di errore: "Memoria esaurita". L'ambiente del DOS può essere ampliato mediante il comando DOS SHELL nel file CONFIG.SYS. Ad esempio, il comando:

```
SHELL=C:\DOS\COMMAND.COM /p /e:512
```

inserisce il processore di comando COMMAND.COM in memoria (/p) e imposta le dimensioni massime dell'ambiente, cioè 512 byte (/e:512).

Questa istruzione non impedisce di per se stessa la visualizzazione del messaggio di errore: "Memoria esaurita", dato che QBasic riceve soltanto una copia dell'ambiente attuale e non lo spazio completo allocato nel comando SHELL dell'MS-DOS, e la quantità di memoria utilizzata dalla copia non può essere aumentata. Prima di eseguire il programma QBasic, è tuttavia possibile sfruttare il comando SET per allocare una variabile "fittizia", come riportato nel seguente esempio:

```
SET ESPANS=123456789012345678901234567890
```

A questo punto il programma è in grado di creare lo spazio disponibile nella propria copia dell'ambiente, liberando lo spazio allocato per questa variabile nel seguente modo:

```
ENVIRON "ESPANS=;"
```

Ora lo spazio è disponibile per le nuove variabili e i nuovi valori di ambiente.

Errori

Se l'ambiente non può essere ampliato fino a includere una nuova variabile (oppure un valore più lungo di una esistente), QBasic visualizza il messaggio di errore: "Memoria esaurita". Se il parametro indicato con ENVIRON non è una stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato modifica la variabile di ambiente PROMPT e visualizza il nuovo prompt e l'ambiente modificato.

```
CLS
PRINT "Ecco le impostazioni dell'ambiente attuale: "
SHELL "SET"                    'visualizza l'ambiente attuale

PRINT "Cambia il prompt: "
ENVIRON "PROMPT=Immettere 'exit' per ritornare a QBasic ===>"
SHELL                 'esegue il processore dei comandi con il nuovo prompt

PRINT "Questo è l'ambiente modificato: "
SHELL "SET"                    'visualizza l'ambiente modificato
```

ENVIRON$

Si consulti anche

ENVIRON, SHELL

Scopo

La funzione ENVIRON$ restituisce il valore della variabile di ambiente DOS specificata.

Sintassi

```
ENVIRON$({variabile | numero})
```

variabile è il nome della variabile di ambiente.
numero è un intero che indica la posizione della variabile desiderata nell'elenco delle variabili di ambiente.

Utilizzo

```
PRINT ENVIRON$("PATH")
```

Visualizza il valore della variabile di ambiente PATH, che indica il percorso di ricerca attuale.

```
env$ = ENVIRON$(3)
```

Assegna alla variabile env$ la variabile di ambiente (e il relativo valore) che si trova al terzo posto nell'elenco.

Descrizione

La funzione ENVIRON$ restituisce il valore della variabile di ambiente specificata. Questa funzione accetta il nome della variabile di ambiente o il numero che ne rappresenta la posizione nell'elenco delle variabili. Dato che ENVIRON$ tiene conto delle lettere maiuscole o minuscole, ENVIRON$("PATH") restituisce il percorso di ricerca, mentre ENVIRON$("path") restituisce una stringa nulla (vuota), a meno che il proprio ambiente non comprenda una variabile denominata path.

Se viene specificato un numero superiore a quello delle variabili di ambiente, ENVIRON$ restituisce una stringa nulla.

Commenti

È necessario tenere presente che l'ambiente esaminato da ENVIRON$ è una copia dell'ambiente DOS che il programma riceve e risulta diverso da quello originale DOS, se vengono apportate modifiche mediante l'istruzione ENVIRON.

Esempio

Il programma di seguito riportato utilizza innanzitutto ENVIRON$ in un ciclo per elencare tutte le variabili nell'ambiente; si noti che il test per la stringa nulla termina

il ciclo, dopo che l'ultima variabile è stata elencata. La seconda parte del programma accetta un nome di variabile di ambiente da parte dell'utente e quindi ne visualizza il valore, se esiste.

```
CLS
var = 1
DO WHILE (ENVIRON$(var) <> "")
   PRINT var; ": "; ENVIRON$(var)
   var = var + 1
LOOP

DO
   PRINT : PRINT "Immettere 'u' per terminare o"
   INPUT "il nome di una variabile da esaminare: "; nomeVar$
   IF UCASE$(nomeVar$) = "U" THEN END
   PRINT nomeVar$; " ";
   IF ENVIRON$(nomeVar$) = "" THEN
      PRINT "non definita"
      ELSE PRINT "è "; ENVIRON$(nomeVar$)
   END IF
LOOP
```

IOCTL

Si consulti anche

IOCTL$

Scopo

L'istruzione IOCTL invia una stringa di istruzioni a un driver di dispositivo; può essere utilizzata per inizializzare un modem oppure per inviare istruzioni al mouse.

Sintassi

```
IOCTL [#] numfile, stringa
```

numfile è il numero di file utilizzato nell'istruzione OPEN che ha aperto il dispositivo come un file.

stringa è una stringa contenente un'istruzione significativa per il driver di dispositivo; deve essere racchiusa tra doppi apici e non superare i 32767 caratteri.

Utilizzo

```
OPEN "SCANNER" FOR OUTPUT AS #4
IOCTL #4, "READPAGE"
```

Apre il driver di dispositivo di uno scanner, in modo che possa ricevere l'output, inviando quindi un comando.

Descrizione

Il DOS supporta qualsiasi driver di dispositivo che accetta le istruzioni IOCTL. Un driver di dispositivo è un programma residente in memoria che traduce le istruzioni generali relative a un dispositivo (ad esempio la stampante, l'unità disco o il mouse) in quelle specifiche che l'hardware può utilizzare. I driver interni, ad esempio LPT1

e COM1, che non supportano le istruzioni IOCTL possono essere sostituti da quelli perfezionati che invece sono in grado di supportarle. Se un driver supporta IOCTL, la documentazione fornita dovrebbe elencare le istruzioni corrette. Si noti che la maggior parte dei driver di dispositivo non supporta le istruzioni IOCTL.

È innanzitutto necessario installare il driver di dispositivo (generalmente utilizzando l'istruzione DEVICE nel file CONFIG.SYS) e quindi aprirlo con un'istruzione OPEN nel proprio programma, prima di poter utilizzare IOCTL per inviare le istruzioni.

Errori

Se il driver specificato non supporta le istruzioni IOCTL, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita".

Esempio

Il programma di seguito riportato illustra l'utilizzo di IOCTL, presupponendo che sia stato installato un driver denominato MYDRIVER.SYS, in grado di gestire questo tipo di istruzioni.

```
OPEN "C:\SYS\MIODRIVE.SYS" FOR OUTPUT AS #1       'apre il driver del
                                                  'dispositivo come un file
IOCTL #1, "BLOCK"                                 'usa la modalità blocco
IOCTL #1, "1024"                'usa una lunghezza del blocco di 1024 byte
IOCTL #1, "READ 4"              'legge quattro blocchi
IOCTL #1, "SEND"                                  'spedisce la data
CLOSE                                             'chiude il driver
```

IOCTL$

Si consulti anche

IOCTL, OPEN

Scopo

La funzione IOCTL$ restituisce una stringa contenente i dati di controllo ricevuti dal driver di dispositivo che supporta le istruzioni IOCTL; può essere utilizzata per ricevere le informazioni relative a un modem o a un mouse.

Sintassi

```
IOCTL$([#] numfile)
```

numfile è il numero di file utilizzato nell'istruzione OPEN che ha aperto il dispositivo come un file.

Utilizzo

```
OPEN "MODEM" FOR OUTPUT AS #1
PRINT IOCTL$(#1)
```

Apre il driver di dispositivo MODEM e visualizza le informazioni di stato restituite.

Descrizione

Un driver di dispositivo è un programma residente in memoria, che traduce le istruzioni generali relative a un dispositivo (ad esempio la stampante, l'unità disco o il mouse) in quelle specifiche che l'hardware può utilizzare. La funzione IOCTL$ può essere utilizzata soltanto con i driver che supportano le istruzioni IOCTL.

Un driver di dispositivo installato, che supporta IOCTL, è in grado di fornire le informazioni di stato nella funzione IOCTL$. Generalmente le informazioni di stato fornite riguardano una stringa con i dati di controllo inviata con l'istruzione IOCTL. Si noti che la maggior parte dei driver di dispositivo non supporta le istruzioni IOCTL.

È innanzitutto necessario installare il driver di dispositivo (generalmente utilizzando l'istruzione DEVICE nel file CONFIG.SYS) e quindi aprirlo con un'istruzione OPEN nel proprio programma, prima di poter utilizzare la funzione IOCTL$.

Errori

Se il driver specificato non supporta le istruzioni IOCTL, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita".

Esempio

Il programma di seguito riportato presuppone che sia stato installato un driver denominato MYDRIVER.SYS; e apre innanzitutto il dispositivo come numero di file 1; se IOCTL$ restituisce 0, il driver non è ancora pronto a ricevere l'input.

```
OPEN "C:\SYS\MIODRIVE.SYS" FOR OUTPUT AS #1      'apre il driver del
                                                 'dispositivo come un file

IF IOCTL$(#1) = 0 THEN         'il dirver restituisce 0 se non abilitato
   PRINT "Driver non abilitato"
   END                         'esce da programma
ELSE
   'utilizza il driver
END IF

CLOSE #1                       'chiude il driver
```

KILL

Si consulti anche

FILES, RMDIR

Scopo

L'istruzione KILL cancella un file e libera lo spazio occupato; può essere utilizzata esattamente come i comandi DEL o ERASE del DOS, ovvero per liberare lo spazio quando il disco è pieno oppure per cancellare i file non necessari.

Sintassi

```
KILL specfile
```

specfile è una variabile stringa o una costante che indica il file o i file da cancellare; può comprendere i caratteri jolly ? e * per indicare un gruppo di file.

Utilizzo

```
KILL "CONFIG.DAT"
```

Cancella il file CONFIG.DAT nella directory corrente.

```
KILL "*.DAT"
```

Cancella tutti i file con l'estensione DAT presenti nella directory in uso.

Descrizione

L'istruzione KILL cancella i file indicati dalla variabile stringa o costante specificata. È anche possibile utilizzare nel nome del file i caratteri jolly ? e *: il primo corrisponde a un carattere qualsiasi nella stessa posizione nel nome del file; il secondo corrisponde a zero o più caratteri qualsiasi nel nome del file.

Se il nome dell'unità disco viene tralasciato, KILL utilizza quello attivo; se invece viene omesso il percorso, KILL utilizza la directory attuale dell'unità disco in uso.

Commenti

I caratteri jolly possono essere utilizzati soltanto nel nome del file, dato che QBasic non è in grado di riconoscerli quando si tratta del nome della directory, dell'unità o del percorso.

Errori

L'istruzione KILL consente di cancellare qualsiasi file chiuso. Se si cerca di cancellarne uno aperto, QBasic visualizza il messaggio di errore: "File già aperto". KILL cancella soltanto i file; per eliminare le directory vuote, è necessario sfruttare l'istruzione RMDIR.

Se le specifiche del file non corrispondono a nessun file, QBasic visualizza il messaggio di errore: "File non trovato". Se viene specificata una lettera di un'unità disco esterna che non contiene un disco, QBasic visualizza il messaggio di errore: "Disco non pronto"; se invece il disco indicato non esiste, viene visualizzato il messaggio: "Dispositivo non disponibile".

Avvertenze

Al contrario del comando DEL del DOS, l'istruzione KILL non visualizza un messaggio quando l'operazione riguarda la cancellazione di tutto il contenuto di una directory. Una sola istruzione KILL è in grado di cancellare tutta una directory, persino il file di programma attualmente in esecuzione; per questa ragione è necessario prestare particolare attenzione, quando la si utilizza nel proprio programma.

Esempio

Il programma di seguito riportato utilizza FILES per visualizzare la directory specificata dall'utente e quindi KILL per cancellare tutti i file in essa contenuti.

```
DO
  PRINT "Immettere il percorso della directory di cui vuoi cancellare i file."
  PRINT "Premi Invio per terminare."
  LINE INPUT specFile$
```

```
    IF specFile$ = "" THEN EXIT DO

    PRINT
    FILES specFile$ + "\*.*"              'visualizza i file nella directory
    PRINT

    LINE INPUT "Vuoi cancellare questi file? (s/n) "; siNo$

    IF UCASE$(siNo$) = "S" THEN
     KILL specFile$ + "\*.*"            'cancella i file nella directory
     PRINT "Tutti i file nella directory "; specFile$; " sono stati cancellati."
     PRINT
     PRINT
    END IF
LOOP UNTIL specFile$ = ""
```

MKDIR

Si consulti anche

CHDIR, RMDIR

Scopo

L'istruzione MKDIR crea la directory specificata e può essere utilizzata per consentire all'utente di creare una nuova directory dall'interno del programma, memorizzandola in un punto qualsiasi su disco.

Sintassi

```
MKDIR directory
```

directory è un'espressione stringa con una lunghezza massima di 127 caratteri, che indica la subdirectory da creare.

Utilizzo

```
MKDIR "C:\IMP\TEMP"
```

Crea TEMP, una subdirectory della directory IMP sul disco C.

Descrizione

L'istruzione MKDIR funziona essenzialmente come l'omonimo comando del DOS, ovvero crea una nuova directory sull'unità disco specificata. MKDIR crea la directory come una subdirectory della directory attiva, a meno che non venga specificato un percorso diverso.

È necessario assicurarsi di poter intercettare gli eventuali errori che si possono verificare con l'istruzione MKDIR, in modo che l'utente possa sapere se una directory non può essere creata perché già esistente o perché esiste un file con lo stesso nome.

L'istruzione MKDIR di QBasic non può essere abbreviata in MD, come con il comando del DOS.

Errori

Se viene specificata una directory già esistente, oppure se un file ha lo stesso nome di una directory, QBasic visualizza il messaggio di errore: "Errore di accesso al percorso di ricerca del file". Se viene indicata un'espressione che non è una stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato richiede all'utente di immettere il nome di una directory e quindi cerca di crearla, stampando un messaggio di errore se la directory è già presente o se esiste già un file con quel nome.

```
ON ERROR GOTO GestErrore                'imposta la gestione degli errori

INPUT "Directory da creare: "; dir$
MKDIR dir$                              'tenta di creare la directory
END

GestErrore:
    numErrore = ERR
    IF numErrore = 75 THEN
       PRINT "Impossibilitato a creare la directory "; dir$
       PRINT "La directory esiste già o "
       PRINT "esiste un file con quel nome. "
    END IF
    RESUME NEXT
```

REM

Si consulti anche

$DYNAMIC, $STATIC

Scopo

L'istruzione REM consente di inserire un commento in un programma; al posto della parola chiave REM, è anche possibile utilizzare un apice. Un commento è un'istruzione non eseguibile che illustra o descrive l'operazione di un programma. REM può inoltre introdurre un metacomando (un'istruzione speciale al compilatore), ad esempio $DYNAMIC.

Sintassi

```
[REM || '] commento
```

commento è una parte di testo qulasiasi. Se viene utilizzata la parola chiave REM, un commento che segue un'istruzione eseguibile sulla stessa riga può iniziare con i due punti, mentre gli spazi prima e dopo i due punti sono facoltativi.

Utilizzo

```
REM Eseguire l'operazione ogni giorno
FOR giorno% = 1 to 7
   CALL Totale(vendite(giorno%))     'richiama il sottoprogramma Totale
NEXT giorno%
```

Il primo commento illustra lo scopo della sezione del codice, mentre il secondo descrive l'azione di una riga.

Descrizione

L'istruzione REM introduce un commento, una descrizione, una spiegazione o un metacomando nel programma. QBasic ignora il testo (tranne nel caso di un metacomando) che segue la parola chiave REM oppure un apice, fino all'inizio della riga successiva. Per aggiungere un commento a un'istruzione eseguibile, è necessario inserire dopo l'istruzione i due punti, iniziando quindi l'istruzione REM, come nel seguente esempio:

```
CALL Albero : REM catalogare il disco fisso
```

Si noti che è anche possibile utilizzare un apice, invece dei due punti e della parola chiave REM, ovvero:

```
CALL Albero ' catalogare il disco fisso
```

In un programma che utilizza i numeri di riga, l'istruzione REM può essere sfruttata come riga di destinazione per le istruzioni GOTO o GOSUB. L'esecuzione prosegue in corrispondenza della prima istruzione eseguibile dopo REM.

Avvertenze

Per inserire un commento all'interno dell'istruzione DATA, è necessario utilizzare REM, invece dell'apice, dato che, in questo caso, QBasic considera l'apice come parte dei dati immessi.

Un commento creato con l'apice non deve essere confuso con una serie di doppi apici, che indica invece una stringa.

Esempio

Il programma di seguito riportato comprende alcuni commenti creati con l'istruzione REM e con un apice.

```
REM questo programma visualizza la stringa "Ciao" 50 volte

FOR i% = 1 TO 50
    PRINT "Ciao";              'visualizza la stringa
NEXT i%

END
```

RMDIR

Si consulti anche

CHDIR, FILES, KILL, MKDIR

Scopo

L'istruzione RMDIR elimina una directory da disco. Per poter essere cancellata, la directory deve essere vuota.

Sintassi

```
RMDIR directory
```

directory è un'espressione stringa contenente non più di 127 caratteri, che indica la subdirectory da cancellare.

Utilizzo

```
RMDIR "C:\ACCT\TEMP"
```

Rimuove la subdirectory TEMP dalla directory ACCT sull'unità disco C.

```
RMDIR dir$
```

Elimina la directory specificata dalla variabile stringa dir$.

Descrizione

L'istruzione RMDIR cancella una directory che non contiene alcun file; funziona essenzialmente come il comando RMDIR del DOS, tranne per il fatto che, in QBasic, non è possibile utilizzare l'abbreviazione RD. Se il programma crea una directory per memorizzare i file temporanei che contengono le fasi intermedie di un'elaborazione, è possibile ad esempio sfruttare l'istruzione RMDIR per eliminare tale directory, quando non serve più. A questo scopo, è necessario innanzitutto cancellare i file temporanei mediante l'istruzione KILL, quindi è possibile rimuovere la directory utilizzando RMDIR.

Errori

Se la directory specificata nell'istruzione RMDIR non esiste, QBasic visualizza il messaggio di errore: "Percorso di ricerca non trovato". Se invece la directory indicata è vuota, QBasic visualizza il messaggio: "Errore di accesso al percorso di ricerca del file", che viene presentato anche quando viene indicato il nome di un file, anziché quello di una directory. Se nell'istruzione RMDIR non viene specificata una stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Esempio

Il programma di seguito riportato consente di eliminare una directory; utilizza inoltre l'istruzione ON ERROR GOTO per intercettare il messaggio: "Errore di accesso al percorso di ricerca del file", che generalmente indica che la directory contiene ancora dei file; il gestore degli errori richiede all'utente se desidera cancellare i file. Si noti che il programma non distingue tra una directory non vuota e un file; se l'utente immette il nome di un file, l'istruzione KILL non funziona e QBasic visualizza il messaggio di errore: "Percorso di ricerca non trovato", eliminando quindi la necessità di effettuare una distinzione tra una directory non vuota e un file.

```
ON ERROR GOTO GestErrore                  'imposta la gestione degli errori

INPUT "Directory da rimuovere: "; dir$
RMDIR dir$                                'tenta di rimuovere la directory
END
```

```
GestErrore:
    numErrore = ERR
    PRINT "Numero errore: "; numErrore
    SELECT CASE numErrore
    CASE 75                         'la directory probabilmente non è vuota
         PRINT dir$; " non vuota."
         PRINT "Vuoi ancora distruggerla?"
         PRINT "(Tutti i file nella directory saranno cancellati.)"
         INPUT risp$
         IF UCASE$(risp$) = "S" THEN
            cancDir$ = dir$ + "\*.*" 'costruisce un percorso contenente
                                     'caratteri sostitutivi
            KILL cancDir$            'cancella tutti i file nella directory
            RESUME                   'ritenta la rimozione della directory
         END IF
    CASE 76                            'la directory non è stata individuata
         PRINT dir$; " non trovata."
    CASE ELSE
         ON ERROR GOTO 0               'disabilita la gestione degli errori
    END SELECT
    RESUME NEXT
```

RUN

Si consulti anche

CHAIN, CLEAR, GOTO, SHELL

Scopo

L'istruzione RUN può essere utilizzata per riavviare il programma in uso dall'inizio o partendo dal numero di riga specificato, oppure per eseguire un altro programma. RUN può essere ad esempio sfruttata per riavviare un programma dopo un errore, oppure per eseguire una serie di programmi autonomi, che non devono condividere le variabili.

Sintassi

```
RUN [riga | file]
```

riga è il numero di riga del programma attivo, dalla quale deve ripartire l'esecuzione.
file è un'espressione stringa di non più di 127 caratteri, che contiene il nome del programma da caricare in memoria e da eseguire.

Utilizzo

```
RUN 100
```

Avvia nuovamente il programma, partendo dalla riga 100.

```
RUN "FINEGIOCO"
```

Carica in memoria il programma FINEGIOCO e lo esegue.

Descrizione

L'istruzione RUN trasferisce il controllo al programma attivo (all'inizio o in corrispondenza del numero di riga specificato) oppure a un altro programma, ma non

ritorna al programma che l'ha richiamata. Se viene specificato un numero di riga, RUN agisce in modo analogo all'istruzione GOTO, tranne per il fatto che chiude tutti i file aperti ed elimina tutte le variabili. In effetti, RUN esegue nuovamente il programma attivo, partendo dalla riga specificata. Se invece RUN viene utilizzata senza alcun parametro, il programma in uso viene eseguito dall'inizio. Con questa istruzione, non è possibile utilizzare un'etichetta di riga non numerica come destinazione. L'istruzione RUN può anche essere sfruttata per eseguire un altro programma, partendo dalla prima riga.

Se il programma non si trova nella directory in uso, è necessario indicarne la posizione inserendo il nome del percorso. Dato che il programma specificato deve essere un file sorgente di QBasic, non è necessario indicare l'estensione, infatti QBasic presuppone che sia BAS. Oggigiorno, i programmatori sfruttano raramente RUN per eseguire un secondo programma, dato che l'istruzione CHAIN è decisamente più versatile, visto che consente di eseguire un altro programma, di gestire i file aperti e di accedere a qualsiasi variabile specificata con l'istruzione COMMON, offrendo quindi maggiori vantaggi quando si utilizza un gruppo di programmi correlati.

Commenti

RUN non consente di mantenere o di condividere i valori delle variabili dal programma originale. Inoltre, dato che chiude tutti i file aperti, è necessario tenere presente che il secondo programma potrebbe doverli riaprire.

Errori

Se il programma indicato nell'istruzione RUN non esiste, QBasic visualizza il messaggio di errore: "File non trovato". Se il numero di riga specificato non esiste oppure non si trova nel codice a livello del modulo, QBasic visualizza il messaggio di errore "Etichetta non definita".

Esempio

Il programma di seguito riportato rappresenta il "front end" di un programma applicativo e utilizza RUN per eseguire il programma PLANNER.BAS, dopo aver visualizzato le istruzioni (su richiesta dell'utente).

```
CLS
PRINT "Benvenuto al Progetto Pianificazione."
INPUT "Desideri vedere le istruzioni (s/n)"; risp$

IF UCASE$(risp$) = "S" THEN
   PRINT "Istruzioni per usare PIANO.BAS"
   PRINT "Maggiori informazioni ...": PRINT
   PRINT "Premi un tasto qualsiasi per continuare ..."
   risp$ = INPUT$(1)             'aspetta la pressione di un tasto
END IF
RUN "PIANO.BAS"                  'assume che il file sia nella stessa
directory
END

' PIANO.BAS
CLS
PRINT "*** PIANO - MENU PRINCIPALE ***"
PRINT "La parte rimanente del programma PIANO viene qui eseguita."
```

SHELL

Si consulti anche

CHAIN, ENVIRON, RUN

Scopo

L'istruzione SHELL consente di eseguire un comando del DOS (oppure qualsiasi file batch o eseguibile) da un programma QBasic. In effetti, SHELL aggiunge tutte le capacità del DOS a QBasic. Ad esempio, sebbene non esista alcun comando di QBasic che consenta di formattare un dischetto, è possibile eseguire questa operazione sfruttando SHELL per richiamare il comando FORMAT del DOS.

Sintassi

```
SHELL [comando]
```

comando è un'espressione stringa che contiene un comando che il DOS è in grado di eseguire. Se non viene specificato alcun parametro, SHELL esegue il processore dei comandi. Per ritornare al programma, è sufficiente digitare EXIT al prompt del DOS.

Utilizzo

```
SHELL "FORMAT B:"
```

Formatta il dischetto nell'unità B da un programma QBasic; il comando FORMAT del DOS visualizza i messaggi consueti.

```
cmd$ = "DIR " + file$ + " /p"
```

Esegue la riga di comando del DOS contenuta nella variabile cmd$.

Descrizione

L'istruzione SHELL esegue una copia del processore di comando del DOS, COMMAND.COM, che eredita l'ambiente DOS utilizzato dai programmi QBasic attivi. SHELL equivale quindi al comando COMMAND con il parametro /C. Qualsiasi comando presente nella stringa specificata, subito dopo la parola chiave SHELL, viene passato al DOS, proprio come se fosse stato immesso al prompt.

Se l'istruzione SHELL indica un comando privo di estensione, il DOS ricerca innanzitutto un file di comando con l'estensione COM, quindi un file eseguibile (EXE) e infine un file batch (BAT). A meno che non venga indicato un percorso, il DOS presuppone che il file si trovi nella directory in uso. Tuttavia, se l'ambiente comprende la variabile PATH, il DOS ricerca le directory specificate per quel file.

Anche se il comando del DOS o il programma eseguito con SHELL non hanno accesso diretto alle variabili presenti nel programma QBasic, è possibile inviare informazioni a essi impostando le variabili nell'ambiente, oppure specificando delle opzioni nella stringa di comando relativa a SHELL.

Errori

Dato che **SHELL** passa il controllo al DOS, la maggior parte degli errori (ad esempio: "File non trovato") vengono riportati dal DOS, non da QBasic (si consulti la documentazione fornita con il DOS, per ulteriori spiegazioni sugli errori DOS). Se nell'istruzione **SHELL** viene indicato un parametro che non è una stringa, QBasic visualizza il messaggio di errore: "Mancata corrispondenza dei tipi".

Avvertenze

Dato che un programma QBasic rimane in memoria durante l'esecuzione di **SHELL**, non è possibile eseguire un programma utilizzando **SHELL**, se esso utilizza una quantità di memoria superiore a quella disponibile. In questo caso, il DOS visualizza il messaggio: "Memoria insufficiente".

Esempio

Il programma di seguito riportato fornisce un menu contenente diversi comandi DOS che il programma esegue o richiede. L'utente deve immettere una E per uscire dal programma. Il ciclo più interno (attorno al menu) visualizza il menu, fino a quando l'utente non immette un carattere corretto. Si noti come è stata creata la riga di comando da utilizzare con ogni istruzione **SHELL**; dato che le istruzioni **SHELL "DIR file$"** e **SHELL "DIR"; file$** non sono corrette, è necessario realizzare la stringa di comando concatenando il comando DOS con la stringa immessa dall'utente. Talvolta occorre aggiungere alcuni spazi, ad esempio è necessario aggiungere uno spazio tra il comando COPY e il nome del file sorgente e tra quest'ultimo e il file di destinazione.

```
DO             'ciclo più esterno - cicla fino a quando l'utente preme U
   DO 'ciclo più interno - cicla fino a quando l'utente preme una lettera valida
      CLS
      PRINT "Accesso al DOS da QBasic per: "
      PRINT "V)isualizza un file"
      PRINT "D)irectory"
      PRINT "S)tampare un file"
      PRINT "C)opiare un file"
      PRINT "U)scire dal programma"
      INPUT scelta$
      scelta$ = UCASE$(scelta$)
   LOOP UNTIL INSTR("VDSCU", scelta$) <> 0

   CLS
   SELECT CASE scelta$
   CASE "V"
      INPUT "Visualizzare il file "; file$
      comando$ = "MORE <" + file$
      SHELL comando$
   CASE "D"
      INPUT "Directory "; dir$
      comando$ = "DIR" + dir$ + " /p"
      SHELL comando$
   CASE "S"
      INPUT "Stampa il file "; file$
      comando$ = "PRINT " + file$
      SHELL comando$
   CASE "C"
      INPUT "File da copiare: ", sorgente$
      INPUT "Destinazione della copia: ", destinazione$
```

```
      comando$ = "COPY " + sorgente$ + " " + destinazione$
      SHELL comando$
   CASE "U"
      END
   CASE ELSE
      PRINT "Comando non valido: "; scelta$
   END SELECT

   PRINT : PRINT "Premi un tasto qualsiasi per continuare"
   risp$ = INPUT$(1)
LOOP
```

STOP

Si consulti anche

END

Scopo

L'istruzione STOP termina un programma QBasic.

Sintassi

```
STOP
```

Utilizzo

```
PRINT "Eseguire la prima riga"
STOP
```

Termina il programma, dopo aver eseguito l'istruzione PRINT.

Descrizione

Il comando STOP interrompe l'esecuzione di un programma, lasciando i file aperti e restituendo il controllo all'ambiente QBasic. STOP è analogo a END, tranne per il fatto che lascia i file aperti, mentre END li chiude.

La parola chiave STOP viene anche sfruttata con diverse istruzioni di gestione degli eventi per sospendere l'intercettazione dell'evento specificato.

Esempio

Il programma di seguito riportato esegue tre istruzioni di assegnazione che utilizzano la funzione RND; quando il controllo raggiunge l'istruzione STOP, il programma termina. Si noti che la quarta istruzione di assegnazione non viene eseguita, a meno che non venga selezionato il comando Continua dal menu Esegui per completare il programma.

```
primo = INT(3 * RND) + 1
secondo = INT(3 * RND) + 2
terzo = INT(3 * RND) + 3
STOP
quarto = primo + secondo + terzo
```

SYSTEM

Si consulti anche

END, SHELL, STOP

Scopo

L'istruzione SYSTEM chiude tutti i file aperti e termina un programma QBasic.

Sintassi

```
SYSTEM
```

Utilizzo

```
PRINT "Questo programma è in fase di esecuzione nell'ambiente QBasic"
SYSTEM
```

Restituisce il controllo all'ambiente QBasic.

Descrizione

L'istruzione SYSTEM termina l'esecuzione di un programma QBasic, cancellandolo dalla memoria. Se il programma viene eseguito specificando il suo nome e l'opzione /RUN quando viene avviato QBasic, il controllo ritorna al sistema operativo. Tuttavia, se il programma viene eseguito in ambiente QBasic, il controllo ritorna all'ambiente. Se l'istruzione SYSTEM viene immessa nella finestra Immediato QBasic termina, restituendo il controllo al sistema operativo.

Esempio

Il programma di seguito riportato dimostra che l'istruzione SYSTEM restituisce il controllo all'ambiente da cui viene eseguito il programma. Se quest'ultimo viene eseguito dall'interno dell'ambiente QBasic, il controllo viene restituito a quell'ambiente; se invece il file viene eseguito specificando l'opzione /RUN quando viene avviato QBasic, il controllo ritorna al DOS.

```
PRINT "Il programma è in esecuzione."
PRINT "Il programma è ancora in esecuzione";
FOR i% = 1 TO 10
    PRINT ".";
    SLEEP 1
NEXT i%

SYSTEM
```

Capitolo 19

Porte e memoria

INTRODUZIONE

In questo capitolo viene esaminato il modo in cui i programmi QBasic possono accedere alle informazioni memorizzate nella memoria del computer e viene inoltre illustrato come comunicare direttamente con l'hardware utilizzando la memoria presente aldilà del normale spazio di indirizzamento, ovvero i canali di input/output, conosciuti come *porte.* Nella Tabella 19.1 sono riportate in ordine alfabetico le istruzioni e le funzioni di QBasic relative alla memoria e alle porte.

MEMORIA

Diverse istruzioni e funzioni di QBasic consentono di fare riferimento ai dati memorizzati in corrispondenza di un indirizzo specifico nella memoria del computer. Ad esempio, la funzione PEEK restituisce il contenuto di un byte in una determinata posizione; l'istruzione CALL ABSOLUTE consente di eseguire un programma in linguaggio macchina, che inizia all'indirizzo specificato in memoria. Queste istruzioni richiedono che l'indirizzo sia un numero intero compreso nell'intervallo da 0 a 65535, perciò è possibile accedere a un blocco di memoria che contiene soltanto fino a 64 KB. Ogni blocco di memoria viene definito *segmento*; il segmento dati di QBasic, DGROUP, è un segmento di 32 KB, a cui è possibile accedere.

I PC IBM e compatibili possono avere fino a 640 KB di memoria ad accesso casuale e 384 KB di memoria dedicata all'adattatore video, al controller dell'unità disco e ai programmi BIOS della ROM. Quindi, è possibile utilizzare fino a 1 MB (1024 KB) di memoria convenzionale.

Tabella 19.1 Le istruzioni e le funzioni relative alle porte e alla memoria

Istruzione o funzione	Descrizione
BLOAD	Carica un blocco di memoria salvato in un file in memoria a un indirizzo assoluto.
BSAVE	Salva un blocco di memoria in un file su disco.
FRE	Restituisce la quantità di spazio disponibile nello stack, nello spazio stringa o in memoria.
INP	Restituisce il byte alla porta specificata.
OUT	Invia un byte alla porta specificata.
PEEK	Restituisce il byte all'indirizzo specificato.
POKE	Scrive un byte all'indirizzo specificato.
WAIT	Attende di leggere un determinato segnale dalla porta specificata.

Indirizzamento della memoria

È possibile specificare qualsiasi indirizzo in memoria utilizzando due numeri di 16 bit: il segmento e l'*offset*. Il numero del segmento indica il blocco di memoria a cui accedere, mentre il numero dell'offset specifica la posizione dell'indirizzo, all'interno del blocco di memoria. Dato che il segmento e l'offset sono entrambi numeri di 16 bit, spesso i programmatori scrivono gli indirizzi sotto forma di valori esadecimali di 4 cifre, separati da due punti. Ad esempio, l'indirizzo corrispondente al segmento 15, offset 145, viene espresso come 000F:0091.

Dato che ogni segmento in memoria inizia al confine del paragrafo (un paragrafo è un blocco di 16 byte), 1 MB di spazio di indirizzamento nel PC può essere suddiviso in 65535 segmenti possibili. Ovviamente, qualsiasi segmento superiore ai 16 byte si sovrappone ai segmenti che iniziano in corrispondenza dei confini successivi del paragrafo.

Programmi QBasic in memoria

QBasic utilizza un singolo segmento di 32 KB, DGROUP, per contenere tutte le variabili semplici presenti in un programma. Questo limite di 32 KB è il fattore principale che limita le dimensioni del programma. Soltanto alle variabili complesse, ad esempio gli array dinamici, vengono forniti propri segmenti all'esterno di DGROUP.

Accesso alla memoria

QBasic fornisce alcune istruzioni che consentono di esaminare e modificare direttamente la memoria. La funzione PEEK restituisce il valore del byte nell'indirizzo specificato, mentre l'istruzione POKE permette di impostare un valore nel byte in corrispondenza dell'indirizzo specificato. In entrambi i casi, l'indirizzo è un valore intero senza segno nell'intervallo da 0 a 65535, che corrisponde all'offset del byte,

misurato dall'inizio del segmento in uso. Dato che un byte consiste di 8 bit, POKE può caricare in memoria soltanto valori nell'intervallo da 0 a 255 e PEEK restituisce soltanto valori nello stesso intervallo.

A meno che non venga diversamente specificato, PEEK e POKE accedono a DGROUP, il segmento dati di QBasic. Dato che DGROUP memorizza le stringhe e le variabili semplici, PEEK e POKE consentono di leggere o modificare questi oggetti direttamente, fino a quando si conosce la posizione in cui vengono memorizzati. In effetti, QBasic supporta una funzione che restituisce queste informazioni: VARPTR restituisce l'indirizzo dell'offset di una variabile numerica.

Ovviamente, QBasic offre metodi più semplici per leggere e caricare le variabili in DGROUP, rispetto a PEEK e POKE. Sebbene la prima funzione venga talvolta utilizzata per caricare una routine in codice macchina in una stringa, i vantaggi offerti da queste due funzioni non sono evidenti, finché non vengono utilizzate all'esterno del segmento di dati DGROUP.

Per accedere alla memoria all'esterno di DGROUP è necessario innanzitutto utilizzare l'istruzione DEF SEG per selezionare un nuovo segmento; questa istruzione è in grado di specificare qualsiasi segmento (da 0 a 65535) nello spazio di indirizzamento, e dato che PEEK e POKE possono indirizzare qualsiasi offset all'interno di un segmento completo di 64 KB, è possibile leggere o scrivere i dati in qualsiasi posizione in memoria. Il nuovo segmento selezionato diventa quello in uso per tutte le istruzioni e le funzioni che sfruttano gli indirizzi assoluti di memoria, quali BLOAD, BSAVE, CALL ABSOLUTE, PEEK, e POKE.

Caricamento e salvataggio dei blocchi di memoria

QBasic fornisce due istruzioni che consentono di trasferire i dati tra la memoria e il disco. Se ad esempio il programma utilizza un array per memorizzare i dati, è possibile salvarlo in un file su disco utilizzando il comando BSAVE. Quando il programma viene eseguito la volta successiva, è possibile caricare l'array in memoria utilizzando il comando BLOAD.

Forse l'utilizzo più comune di BLOAD e BSAVE consiste nel caricare e memorizzare le immagini dello schermo. Per BSAVE, il buffer del video è esattamente uguale a qualsiasi altra parte della memoria; questo comando consente di catturare una schermata grafica creata dal programma e di memorizzarla in un file su disco, caricandola in un momento successivo mediante BLOAD.

Il far heap

QBasic memorizza i dati anche in un'area della memoria sopra DGROUP; quest'area, denominata *far heap*, contiene variabili di ampie dimensioni, quali array dinamici numerici e array dinamici di stringhe a lunghezza fissa. Il far heap utilizza la memoria restante dopo che tutti gli altri segmenti sono stati allocati, fino a un massimo di 128 KB.

La funzione FRE consente di esaminare l'utilizzo della memoria da parte del programma. Se viene specificato il valore -1 come parametro, FRE restituisce il numero di byte disponibili nello heap, che è l'area di memoria utilizzata per contenere tutte le

variabili (lo heap viene suddiviso in due parti: il *near heap* è la parte di memoria che contiene le stringhe e le variabili semplici, il far heap memorizza invece gli array numerici e quelli di stringhe a lunghezza fissa). Se viene specificato il parametro -2, FRE restituisce il numero dei byte disponibili nello stack del programma; se invece viene utilizzato un numero qualsiasi o una stringa come parametro, la funzione restituisce lo spazio disponibile per le stringhe. Se viene specificata una stringa, FRE compatta lo spazio stringa disponibile in un singolo blocco di memoria, prima di restituire la quantità di spazio disponibile.

PORTE

Il computer dispone, oltre che della memoria che contiene i dati e i programmi attivi (compreso il DOS e il ROM BIOS), di una parte ulteriore di memoria, a cui è possibile accedere da QBasic. Questo è il blocco di memoria di 64 KB utilizzato dalle porte hardware, che si trova all'esterno del normale spazio di indirizzamento, ma che può essere in ogni caso letto e scritto dai programmi.

Quando si effettuatano comunicazioni mediante una porta, l'operazione di lettura e di scrittura viene eseguita da e in questo blocco di memoria. La funzione INP e l'istruzione OUT sono analoghe a PEEK e POKE, con l'eccezione che prima di accedere all'indirizzo della porta, inviano un segnale sul bus del sistema per indicare che l'indirizzo non è compreso nel normale spazio di indirizzamento del PC, ma è il numero di una porta.

Una porta è un canale di input/output mediante il quale un dispositivo hardware invia informazioni al programma e riceve in cambio istruzioni. I dispositivi hardware comprendono i chip di supporto del processore, la tastiera, il controller del disco, l'adattatore video, il coprocessore matematico e così via. Ogni dispositivo del sistema è associato con una o più porte; dato che gli indirizzi di porta disponibili sono 65536, molti non vengono utilizzati. Nella Tabella 19.2 sono elencati gli indirizzi delle porte e i dispositivi hardware più comuni.

L'istruzione OUT scrive un byte di informazioni nell'indirizzo della porta; la funzione INP restituisce l'impostazione attiva della porta. L'istruzione WAIT attende di leggere un determinato segnale da una porta. Se il computer dispone di un dispositivo di acquisizione dei dati, ad esempio un lettore di codici a barre, l'istruzione WAIT può sospendere l'esecuzione, fino a quando questo non è pronto a trasmettere i dati. La funzione INP e le istruzioni OUT e WAIT vengono raramente utilizzate nella normale programmazione, dato che le istruzioni e le funzioni di QBasic di livello più alto consentono di controllare indirettamente i dispositivi del sistema. Nondimeno, queste istruzioni consentono di programmare l'hardware al livello più basso, ogniqualvolta ciò è necessario.

Bibliografia

Davies, Russ. *Mapping the IBM PC and PCjr.* Greensboro, N.C.; Compute!Books, 1988

Norton, Peter, and Richard Wilton. *The New Peter Norton Programmer's Guide to the IBM PC & PS/2.* 2d ed. Redmond, Wash.: Microsoft Press, 1988.

Tabella 19.2 Gli indirizzi principali delle porte di I/O sui PC

Indirizzo della porta	Descrizione della periferica
000H-01FH	Controller DMA (Direct memory access).
020H-03FH	Controller programmabile dell'interrupt.
040H-05FH	Timer programmabile.
060H-06FH	Controller tastiera.
070H-07FH	Orologio (PC/XT e PS/2 Modello 30) RAM CMOS per la configurazione (PC/AT e altri PS/2).
0A0H	Registro maschera NMI (PC, PC/XT e PS/2 Modello 30).
0C0H-0CFG	Generatore suono di alta qualità (PCJr e Tandy 1000).
0E0H-0EFH	Orologio (PC/AT e serie PS/2).
0F0H-0FFH	Coprocessore matematico.
1F0H-1F8H	Controller disco fisso.
200H-207H	Controller joystick.
278H-27FH	Terza porta parallela.
2F8H-2FFH	Seconda porta seriale.
300H-31FH	Scheda prototipo.
320H-32FH	Controller disco fisso (PC/XT e PS/2 Modello 30).
360H-363H	Controller di rete.
368H-36BH	
378H-37FH	Seconda porta parallela.
3A0H-3A9H	Adattatore bisincrono (collegamento mainframe).
3B0H-3BBH	MDA (Monochrome Display Adapter); utilizzato anche da EGA e VGA nelle modalità video monocromatiche.
3BCH-3BFH	Prima porta parallela.
3C0H-3CFH	Adattatori grafici avanzati (EGA e VGA).
3D0H-3DFH	Adattatore grafici a colori (CGA e MCGA); utilizzati anche da EGA e VGA nelle modalità video a colori.
3F0H-3F7H	Controller Floppy disk.
3F8H-3FFH	Prima porta seriale.

BLOAD

Si consulti anche

BSAVE, DEF SEG, VARPTR, VARSEG

Scopo

L'istruzione BLOAD carica un file *memory-image* (un file che contiene i dati presenti in memoria) creato dall'istruzione BSAVE nella memoria in corrispondenza della posizione specificata. Dato che questa istruzione consente di caricare i file più velocemente rispetto a qualsiasi altro metodo, può essere utilizzata per caricare le immagini dello schermo che non possono essere create velocemente dalle istruzioni grafiche di QBasic.

Sintassi

```
BLOAD nome[, offset]
```

nome è il nome del dispositivo o del file che contiene il memory-image, può comprendere il percorso della directory e il nome dell'unità e può essere un letterale stringa racchiuso tra doppi apici o un'espressione stringa.

offset è l'indirizzo nel segmento attivo in corrispondenza del quale il file viene caricato; è un'espressione numerica nell'intervallo da 0 a 65535. Se questo parametro viene omesso, il file viene caricato nel segmento e nell'indirizzo da cui è stato salvato.

Utilizzo

```
DEF SEF = libSeg%
BLOAD mcFile$, $H400
```

Seleziona il segmento di memoria specificato nella variabile intera libSeg% e quindi carica un file memory-image nell'offset 1024 (400H) in quel segmento.

Descrizione

QBasic presenta due comandi che consentono di trasferire i file memory-image tra il disco e la memoria. Questo tipo di file contiene dati binari, una copia esatta bit per bit di un blocco di memoria che può contenere qualsiasi dato in memoria, ad esempio un array di dati o un'immagine dello schermo. L'istruzione BSAVE salva questo file memory-image su disco, mentre BLOAD lo carica nuovamente in memoria. Anche se, per default, BLOAD restituisce la copia nell'area della memoria da cui è stata salvata, è anche possibile caricare la copia in qualsiasi altra posizione. Dato che BLOAD utilizza una lettura binaria semplice per trasferire tutto il file dal disco nella memoria, questo metodo di caricamento risulta più rapido rispetto ad altri.

L'intestazione del file BLOAD

I file caricati dall'istruzione BLOAD utilizzano un formato di intestazione comune che inizia sempre con il numero esadecimale &HFD (253 decimale), che identifica un file come memory-image. I 4 byte successivi contengono il segmento e l'indirizzo dell'offset del blocco di memoria da cui è stato salvato il file; se l'istruzione BLOAD viene utilizzata senza specificare un offset, QBasic utilizza questo indirizzo per ripristinare il file in memoria. Tuttavia, se viene specificato un valore di offset, BLOAD carica il file partendo da quel parametro, nel segmento specificato dall'ultima istruzione DEF SEG. Gli ultimi 2 byte dell'intestazione contengono la lunghezza del file, in byte. Nella Figura 19.1 è riportato il formato dell'intestazione del file memory-image.

Il memory-image inizia immediatamente dopo l'intestazione del file e viene terminato dal carattere CTRL+Z (ASCII 26), che è il contrassegno standard di fine file del DOS. Dato che vengono forniti soltanto 2 byte per contenere la lunghezza del file, le dimensioni massime di un'immagine che può essere salvata in un solo file sono di 64 KB, oppure di un segmento completo.

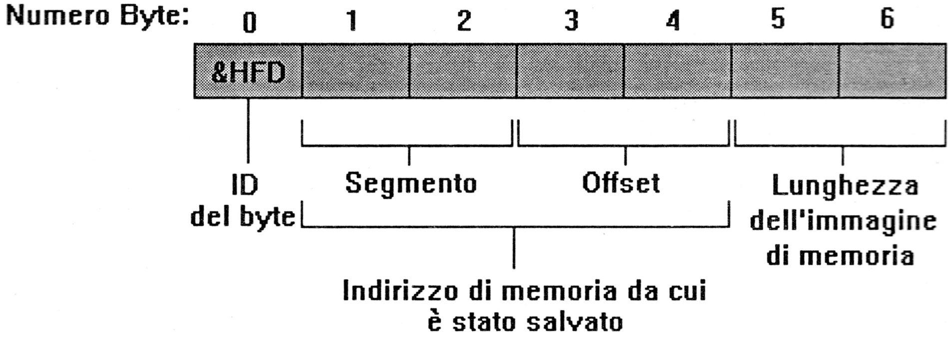

Figura 19.1 Il formato di intestazione dei file memory-image

Avvertenze

Dato che QBasic non verifica che gli indirizzi specificati nell'istruzione BLOAD siano liberi e possano quindi essere utilizzati dal programma, il file di immagine caricato potrebbe sovrascrivere gli altri programmi e persino il sistema operativo.

Quando vengono caricate immagini grafiche, è necessario assicurarsi che la modalità video attiva sia identica a quella in cui è stata salvata l'immagine; in caso contrario, la visualizzazione non risulta corretta.

Esempio

Per caricare le immagini grafiche CGA precedentemente salvate, è sufficiente utilizzare BLOAD con il nome del file da ricaricare; il comando BSAVE ha memorizzato il segmento e l'indirizzo dell'offset del buffer dello schermo nell'intestazione del file. Risulta invece più difficile caricare le schermate EGA e VGA, dato che ognuna di queste richiede quattro file BSAVE, uno per ogni piano di bit utilizzato da questi adattatori. Il sottoprogramma di seguito riportato utilizza BLOAD per caricare i quattro file che costituiscono una schermata grafica a colori EGA o VGA.

I quattro piani di bit utilizzati dagli adattatori EGA e VGA (blu, verde, rosso e alta intensità) vengono assegnati allo stesso segmento della memoria video; controllano l'aspetto dei pixel sulla schermata grafica.

Viene utilizzata la porta all'indirizzo 3C4H, ovvero il registro dell'indirizzo EGA e VGA. Il programma scrive innanzitutto il valore 2 nella porta, per consentire al registro dei dati alla porta 3C5H di impostare il piano di bit attualmente indirizzabile, selezionando quindi un piano di bit alla volta e scrivendo i valori 1, 2, 4 e 8 nella porta 3C5H.

```
SUB LeggiVGA (titolo$) STATIC
    SHARED modoSchermo AS INTEGER

    IF LEN(titolo$) > 8 THEN titolo$ = LEFT$(titolo$, 8)
    IF modoSchermo% > 6 THEN
       IF modoSchermo% = 13 THEN
          BLOAD titolo$ + ".GRA"        'legge il solo piano di bit
       ELSE
```

```
            FOR i% = 0 TO 3
                OUT &H3C4, 2            'abilita la porta
                OUT &H3C5, 2 ^ i%       'imposta il piano di bit indirizzabile
                                        'correntemente
                BLOAD titolo$ + ".BP" + CHR$(49 + i%)'legge il piano di bit
                IF i% = 1 AND modoSchermo% = 10 THEN EXIT FOR
            NEXT i%
        END IF
    END IF
END SUB
```

BSAVE

Si consulti anche

BLOAD, DEF SEG, SCREEN (Istruzione), VARPTR, VARSEG

Scopo

L'istruzione BSAVE trasferisce un file memory-image su disco o su un dispositivo; può essere utilizzata per salvare immagini complesse dello schermo, create dalle istruzioni grafiche.

Sintassi

```
BSAVE nome, offset, byte
```

nome è un'espressione stringa che contiene il nome del dispositivo o del file in cui deve essere salvato il file memory-image; può comprendere il percorso della directory o il nome dell'unità disco.

offset è l'indirizzo, nel segmento attivo, in cui inizia il blocco di memoria da salvare; byte e la lunghezza del blocco di memoria. Questi parametri sono espressioni numeriche nell'intervallo da 0 a 65535.

Utilizzo

```
DEF SEG = &HB800
BSAVE "FIGURA.DAT", 0, 16384
```

Seleziona il segmento video per i computer che utilizzano un adattatore a colori e scrive un'immagine dello schermo di 16 KB nel file FIGURA.DAT. L'operazione di copia inizia in corrispondenza dell'offset 0 del segmento.

Descrizione

BSAVE deve essere utilizzata per trasferire una copia esatta di un'area della memoria in un file memory-image su disco. Questo tipo di file contiene dati binari, una copia esatta bit per bit di un blocco di memoria che può contenere qualsiasi dato in memoria, ad esempio un array di dati o un'immagine dello schermo. È necessario indicare il nome del file in cui si desidera salvare l'immagine, l'indirizzo di partenza e il numero di byte da trasferire. Se il file non esiste ancora, BSAVE lo crea. Per caricare nuovamente il file in memoria, è sufficiente utilizzare BLOAD.

Nell'istruzione BSAVE deve essere specificato soltanto l'offset dell'indirizzo in cui si desidera che inizi la copia. Per copiare da un segmento diverso da quello di default

DGROUP, è necessario innanzitutto modificare i segmenti richiamando l'istruzione DEF SEG. Per eseguire l'operazione di salvataggio partendo dall'inizio del segmento, è sufficiente utilizzare un offset di 0.

La somma dell'offset specificato e del numero di byte non deve essere superiore a 65535 (il numero di byte in un segmento). In caso contrario, QBasic non visualizza un messaggio di errore, ma quando BSAVE raggiunge la fine del segmento, riprende a copiare i byte partendo dall'inizio.

BSAVE risulta utile per salvare le visualizzazioni sullo schermo, ma è valida anche con la modalità testo e i grafici ad alta risoluzione. Per utilizzare BSAVE a questo scopo, è necessario conoscere l'indirizzo del segmento della memoria video utilizzata dall'adattatore e le dimensioni (in byte) dell'immagine nelle diverse modalità video. Nella Tabella 19.3 sono riportate queste informazioni.

Le modalità video utilizzabili soltanto con gli adattatori EGA e VGA supportano uno schema diverso di indirizzamento della memoria video che utilizza i piani di bit.

Tabella 19.3 L'indirizzo e le dimensioni del buffer video nelle modalità video di QBasic

Modalità video	Segmento	Dimensioni pagina (in byte)	Numero di pagine	Osservazioni
0	B000H	4096	1	MDA e HGC
	B800H	4096	4	80 colonne di testo
	B800H	2048	8	40 colonne di testo
1	B800H	16384	1	320 per 200, quattro colori
2	B800H	16384	1	640 per 200, due colori
3	B800H	32768	1	soltanto HGC
11	B800H	65536	1	VGA e MCGA
13	B800H	65536	1	VGA e MCGA

Nota: I computer che dispongono di un adattatore CGA e di un video monocromatico utilizzano l'indirizzo del segmento del buffer B800H, non B000H.

Esempio

L'operazione di salvataggio delle immagini create nelle modalità video utilizzate soltanto dagli adattatori EGA e VGA risulta complicata, dato che BSAVE deve creare quattro file, uno per ogni piano di bit che questi adattatori utilizzano. Il sottoprogramma di seguito riportato estrae ogni piano di bit dal buffer video e quindi utilizza BSAVE per salvarlo in un file. I quattro piani di bit vengono quindi assegnati allo stesso segmento della memoria video.

Questo programma utilizza l'indirizzo di porta 3CEH, ovvero la porta EGA e VGA; scrive innanzitutto il valore 4 nella porta, per consentire al registro dei dati alla

porta 3CFH di impostare il piano di bit attualmente indirizzabile, selezionando quindi un piano di bit alla volta e scrivendo i valori da 1 a 3 nella porta 3CFH.

```
'sottoprogramma per memorizzare schermate EGA e VGA dalla memoria video

SUB SalvaVGA (titolo AS STRING) STATIC
    SHARED modoSchermo AS INTEGER                   'modalità video corrente
    IF LEN(titolo$) > 8 THEN
       titolo$ = LEFT$(titolo$, 8)
    END IF
    SELECT CASE modoSchermo%                        'dimensioni del buffer video
        CASE 7
             byte& = 8000
        CASE 8
             byte& = 16000
        CASE 9, 10
             byte& = 28000
        CASE 11, 12
             byte& = 38400
        CASE 13
             byte& = 64000
        CASE ELSE
             byte& = 0
    END SELECT

    IF byte& THEN
       DEF SEG = &HA000                             'segmento del buffer video
       IF modoSchermo% = 13 THEN
          BSAVE titolo$ + ".GRA", 0, byte&
       ELSE
          FOR i% = 0 TO 3
              OUT &H3CE, 4               'abilita la porta
              OUT &H3CF, i%              'imposta il piano di bit indirizzabile
                                         'correntemente
              BSAVE titolo$ + ".BP" + CHR$(49 + i%), 0, byte&
              IF i% = 1 AND modoSchermo% = 10 THEN EXIT FOR
          NEXT i%
       END IF
       DEF SEG
    END IF
END SUB
```

FRE

Si consulti anche

DIM, $DYNAMIC, ERASE, $STATIC

Scopo

La funzione FRE restituisce la quantità di spazio sullo stack, lo spazio stringa o la memoria disponibile; può essere utilizzata per determinare se il programma dispone di spazio sufficiente per eseguire un compito che richiede molta memoria.

Sintassi

```
FRE (numero | stringa)
```

Se numero è -1, la funzione FRE restituisce la quantità di memoria disponibile nello heap; se il valore è -2, FRE restituisce lo spazio inutilizzato dello stack; per qualsiasi altro

valore numerico restituisce invece le dimensioni del successivo blocco libero per la memorizzazione delle stringhe. Se viene specificato stringa, FRE compatta innanzitutto lo spazio inutilizzato per le stringhe nel blocco più ampio possibile e quindi restituisce la quantità di spazio disponibile.

Utilizzo

```
IF FRE(-2) < 256 THEN PRINT "Lo spazio dello stack si sta esaurendo!"
```

Verifica lo spazio disponibile nello stack e avverte l'utente se è inferiore a 256 byte.

Descrizione

La funzione FRE restituisce il numero di byte disponibili in una delle tre aree di memoria utilizzate da QBasic per memorizzare i dati dei programmi: lo heap, lo stack e lo *spazio stringa*. Lo heap, che è la memoria utilizzata da QBasic per contenere tutte le variabili, viene suddiviso in due parti: il near heap è la parte di memoria che contiene le stringhe e le variabili semplici; il far heap memorizza invece gli array numerici e quelli di stringhe a lunghezza fissa). Lo stack memorizza le variabili temporanee e gli indirizzi di ritorno delle subroutine e delle funzioni. Lo spazio stringa è invece la parte del near heap che contiene i valori delle variabili che comprendono le stringhe.

Il valore restituito da FRE dipende dal parametro specificato nella chiamata alla funzione: se è -1, FRE restituisce la quantità di memoria disponibile nello heap, se viene specificato il parametro -2, FRE restituisce il numero dei byte disponibili nello stack del programma, se viene utilizzato un numero qualsiasi o una stringa come parametro, la funzione restituisce lo spazio stringa disponibile.

Se viene specificata una stringa, FRE compatta lo spazio stringa disponibile in un singolo blocco di memoria, prima di restituire la quantità di spazio disponibile. Questa operazione consente di eliminare le stringhe che non sono più utilizzate e di combinare tutte le parti frammentate.

Si noti che FRE con il parametro -2 (per lo spazio disponibile sullo stack) restituisce risultati affidabili soltanto quando viene richiamata da un programma in esecuzione. Anche se l'istruzione FRE(-2) può essere eseguita dalla finestra Immediato, i risultati ottenuti non sono precisi.

Esempio

Il programma di seguito riportato consente di dimensionare diversi tipi di array e quindi utilizza la funzione FRE per esaminare la quantità di memoria disponibile in base alle modifiche apportate.

```
REM $DYNAMIC              'rende tutti gli array dinamici
DECLARE SUB MemDisponibile ()

CLS
MemDisponibile            'mostra la memoria disponibile
DIM array%(15000)         'array dinamico di interi
PRINT "Dopo la definizione di un array dinamico di 15000 interi:"
MemDisponibile

DIM strArray$(1000)       'array dinamico di stringhe a lunghezza variabile
PRINT "Dopo la definizione di un array dinamico di 1000 stringhe a lunghezza
variabile:"
MemDisponibile
```

```
FOR i% = 1 TO 1000
    strArray$(i%) = "Stringa di prova"
NEXT i%
PRINT "Dopo il riempimento dell'array:"
MemDisponibile

DIM dinString(1000) AS STRING * 10      'array dinamico di stringhe a
lunghezza fissa
PRINT "Dopo la definizione di un array dinamico di 1000 stringhe a lunghezza
fissa:"
MemDisponibile

FOR i% = 1 TO 1000
    dinString(i%) = "Prova"
NEXT i%
PRINT "Dopo il riempimento dell'array:"
MemDisponibile

REM $STATIC

SUB MemDisponibile
    PRINT "Spazio totale disponibile nello heap: "; FRE(-1)
    PRINT "Spazio totale disponibile per le stringhe: "; FRE(0)
    PRINT
END SUB
```

INP

Si consulti anche

OUT, WAIT

Scopo

La funzione INP consente al programma di leggere un byte di dati da una porta specificata.

Sintassi

```
INP(porta)
```

porta è l'indirizzo della porta specificata ed è un numero intero nell'intervallo da 0 a 65535 (o da 0 a FFFF in esadecimale).

Utilizzo

```
IF INP(&H3D5) <> 255 THEN PRINT "L'adattatore è stato installato"
```

La porta 981 (3D5H) è il registro di stato utilizzato dagli adattatori CGA, EGA e VGA nelle modalità colore. Se il valore restituito è 255, è installato un video monocromatico.

Descrizione

Il processore centrale del PC comunica con i chip di supporto e i dispositivi hardware mediante una serie di porte, che sono i canali di input/output attraverso i quali passano le informazioni; QBasic associa una o più porte hardware con ogni dispositivo e presenta le funzioni e le istruzioni necessarie a gestire la maggior parte

delle operazioni riguardanti l'hardware. In ogni caso, se si desidera programmare il proprio adattatore video a basso livello, oppure se il computer utilizza uno scanner digitale o un altro dispositivo non standard, INP e OUT consentono di controllare tale dispositivo senza per questo dover utilizzare una routine in linguaggio macchina.

I PC IBM e compatibili allocano gli indirizzi di porta da 0 a 3FFH nell'hardware di sistema; tuttavia, non tutti questi indirizzi sono poi effettivamente utilizzati. Inoltre, dato che molte porte sono a sola scrittura, INP non sempre restituisce dati significativi. Nella Tabella 19.2 sono riportati gli indirizzi di porta associati ai dispositivi di sistema più importanti.

Dato che INP legge i dati 1 byte alla volta, ogni chiamata a questa funzione restituisce un valore nell'intervallo da 0 a 255. Molte porte di input richiedono di scrivere innanzitutto in un'altra porta associata alla dispositivo, in modo che sia possibile selezionare quale voce dei dati viene restituita. Ad esempio, è possibile caricare una coppia di numeri casuali a 8 bit, scrivendo il valore 0 nella porta di controllo del timer di sistema all'indirizzo 43H e leggendo quindi due volte la porta del timer (all'indirizzo 40H):

```
OUT &H43, 0
cas1% = INP(&H40): cas2% = INP(&H40)
```

Dato che i due valori restituiti da INP in questo esempio sono il byte superiore e quello inferiore di un numero singolo a 16 bit, è necessario leggere la porta due volte per caricare tutto il numero.

Avvertenze

Dato che gli indirizzi di porta sono specifici dell'hardware di cui si dispone, l'utilizzo di INP e OUT limita la portabilità dei propri programmi ai computer che hanno una configurazione hardware analoga. Inoltre, dato che queste istruzioni gestiscono direttamente l'hardware di sistema, se in una porta vengono scritti dati errati il computer potrebbe bloccarsi.

Esempio

I computer della serie PC/AT e PS/2 memorizzano i dati di configurazione nella RAM, che è localizzata in circuiti CMOS e può essere letta mediante la porta hardware 71H. Il programma di seguito riportato legge le informazioni di configurazione e restituisce il numero e il tipo di dischi fissi installati nel sistema. Si noti che il programma scrive un byte all'indirizzo 70H, in modo che il byte letto dalla funzione INP contenga le informazioni necessarie.

```
CLS
OUT &H70, &H12          'specifica l'informazione richiesta
tipoDisco% = INP(&H71)  'legge i dati dal disco fisso
PRINT tipoDisco%
cDrive% = tipoDisco% \ 16   'il drive C è specificato nei primi quattro bit
dDrive% = tipoDisco% MOD 16 'il drive D è specificato nei secondi quattro bit

IF cDrive% THEN
   PRINT "Il disco fisso C è di tipo"; cDrive%
ELSE
   PRINT "Il disco fisso C non è installato"
END IF
```

```
IF dDrive% THEN
   PRINT "Il disco fisso D è di tipo"; dDrive%
ELSE
   PRINT "Il disco fisso D non è installato"
END IF
END
```

OUT

Si consulti anche

INP, WAIT

Scopo

L'istruzione OUT invia un byte di dati a una porta specificata.

Sintassi

```
OUT porta, byte
```

porta è l'indirizzo della porta specificata ed è un numero intero nell'intervallo da 0 a 65535 (o da 0 a FFFF in esadecimale).

byte è un valore nell'intervallo da 0 a 255.

Utilizzo

```
OUT &H70, &H12
```

Scrive il byte &H12 nell'indirizzo della porta CMOS RAM. Questo consente di preparare la porta, in modo che sia in grado di ottenere un determinato valore con la funzione INP.

Descrizione

Una porta è un canale di input/output attraverso il quale un programma può direttamente controllare un dispositivo hardware. Generalmente non è necessario accedere alle porte, dato che QBasic fornisce le funzioni e le istruzioni necessarie a gestire la maggiore parte delle operazioni riguardanti l'hardware. In ogni caso, se si desidera programmare il proprio adattatore video a basso livello, oppure se il computer utilizza uno scanner digitale o un altro dispositivo non standard, OUT e INP consentono di controllare tale dispositivo, senza per questo dover utilizzare una routine in linguaggio macchina.

I PC IBM e compatibili allocano gli indirizzi di porta da 0 a 3FFH nell'hardware di sistema; tuttavia, non tutti questi indirizzi sono poi effettivamente utilizzati. Inoltre, dato che molte porte sono a sola lettura, non è sempre possibile sfruttare OUT. Nella Tabella 19.2 sono riportati gli indirizzi di porta associati ai dispositivi di sistema più importanti. Dato che OUT scrive i dati 1 byte alla volta, il valore inviato alla porta deve essere compreso nell'intervallo da 0 a 255.

Avvertenze

Dato che gli indirizzi di porta sono specifici dell'hardware di cui si dispone, l'utilizzo di **INP** e **OUT** limita la portabilità dei propri programmi ai computer che hanno una configurazione hardware analoga. Inoltre, dato che queste istruzioni gestiscono direttamente l'hardware di sistema, se in una porta vengono scritti dati errati il computer potrebbe bloccarsi.

Esempio

Se gli adattatori EGA e VGA hanno memoria sufficiente, è possibile memorizzare più di una schermata di informazioni alla volta, utilizzando quindi l'istruzione **SCREEN** per passare tra le pagine video. Anche l'hardware stesso consente un controllo più accurato della visualizzazione; in effetti, programmando direttamente i registri di indirizzo dell'adattatore è possibile specificare la posizione della parte visibile del buffer video al pixel più vicino. Il sottoprogramma di seguito riportato traccia una disegno sullo schermo e quindi consente all'utente di premere un tasto di direzione per spostarne il contenuto in qualsiasi punto. Viene utilizzata l'istruzione **OUT** per far scorrere le schermate.

```
DECLARE SUB PanVideo (x%, y%)

CONST FALSO = 0, VERO = NOT FALSO

SCREEN 9
LINE (0, 0)-(639, 349), 9, BF              'traccia il disegno
VIEW SCREEN (40, 25)-(600, 325), 0, 15
CIRCLE (319, 174), 150, 14: PAINT (319, 174), 14, 14

'inizializza le variabili
x% = 0: y% = 0
esc$ = CHR$(27)
sinDirezione$ = CHR$(0) + CHR$(75): desDirezione$ = CHR$(0) + CHR$(77)
suDirezione$ = CHR$(0) + CHR$(72): giuDirezione$ = CHR$(0) + CHR$(80)
DO
  DO
     TastoPremuto$ = INKEY$
  LOOP WHILE TastoPremuto$ = ""
  pan% = VERO
  SELECT CASE TastoPremuto$                    'calcola la nuova coordinata x o y
      CASE sinDirezione$
         IF x% > 0 THEN x% = x% - 1
      CASE desDirezione$
         IF x% < 79 THEN x% = x% + 1
      CASE suDirezione$
         IF y% > 0 THEN y% = y% - 1
      CASE giuDirezione$
         IF y% < 22 THEN y% = y% + 1
      CASE ELSE
         pan% = FALSO
  END SELECT
  IF pan% THEN PanVideo x%, y% * 5               'muove la visualizzazione
LOOP UNTIL TastoPremuto$ = esc$
END

'imposta le coordinate della finestra di visualizzazione

SUB PanVideo (x%, y%) STATIC
    OUT &H3D4, 12: OUT &H3D5, y%           'provoca uno scroll verticale
    OUT &H3D4, 13: OUT &H3D5, x%           'provoca uno scroll orizzontale
END SUB
```

PEEK

Si consulti anche

DEF SEG, POKE, VARPTR

Scopo

La funzione PEEK restituisce il contenuto dell'indirizzo di memoria specificato nel segmento attivo; può essere utilizzata ad esempio per determinare la configurazione hardware del sistema.

Sintassi

```
PEEK(indirizzo)
```

indirizzo è un'espressione di tipo intero nell'intervallo da 0 a 65535, che indica l'offset del byte da esaminare nel segmento di memoria.

Utilizzo

```
byte% = PEEK(&HF000)
```

Assegna il contenuto dell'indirizzo F000H alla variabile di tipo intero byte%. Gli indirizzi possono essere specificati sia nella notazione esadecimale che in quella decimale.

Descrizione

La funzione PEEK consente di esaminare direttamente i dati in memoria. Questi dati potrebbero essere variabili utilizzate dal programma, caratteri sullo schermo o informazioni fornite dal BIOS o dal DOS. L'istruzione DEF SEG può essere utilizzata per specificare il segmento di memoria che si desidera leggere. Se non viene definito un nuovo segmento, QBasic utilizza DGROUP, quello di default.

PEEK restituisce un valore compreso nell'intervallo da 0 a 255, il contenuto di un singolo byte di memoria. Se si desidera leggere un valore diverso da un singolo carattere, è necessario leggere il contenuto di due o più indirizzi di memoria vicini. Ad esempio, un intero viene memorizzato in 2 byte adiacenti, con il byte meno significativo (LSB) che precede quello più significativo (MSB); quest'ultimo indica il numero di volte in cui il valore intero può essere diviso per 256, mentre LSB è il resto di questa divisione. Il numero 1000, ad esempio, verrebbe memorizzato in memoria nel modo riportato nella Figura 19.2.

Quindi, per ottenere il valore di un intero che inizia all'indirizzo specificato dalla variabile loc%, è necessario utilizzare la seguente istruzione:

```
ValoreInt% = PEEK(loc%) + PEEK(loc% + 1) * 256)
```

Esempio

Un programma che deve essere utilizzato su computer diversi, potrebbe dover determinare il tipo di hardware del sistema. Queste informazioni possono essere facilmente reperite su un PC IBM compatibile, dato che in questi computer il ROM

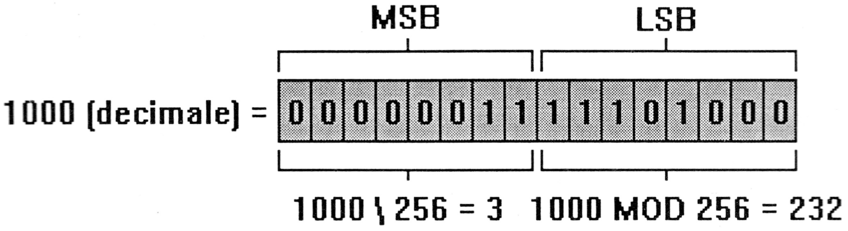

Figura 19.2 Il numero 1000 in memoria

BIOS mantiene un "elenco degli equipaggiamenti" di 2 byte nella memoria bassa. Il programma di seguito riportato utilizza PEEK per leggere questo elenco e quindi visualizzarne il contenuto.

```
DECLARE FUNCTION SInt% (num&)
DECLARE FUNCTION BitTest% (numero%, bit%)
DEF SEG = &H40                           'cambia il segmento di default
                                         'in un'area bassa di memoria
infByte = PEEK(&H10)                     'ottiene i byte meno significativi
                                         'della lista degli equipaggiamenti
supByte = PEEK(&H11)                     'ottiene i byte più significativi
                                         'della lista degli equipaggiamenti
equipagg% = SInt%(infByte + (suByte * 256)) 'forma la lista degli
                                            'equipaggiamenti
DEF SEG                                  'cambia il segmento di default
                                         ' in DGROUP
interruttore$ = STRING$(16, "0")
FOR i% = 15 TO 0 STEP -1
    IF BitTest%(equipagg%, i%) THEN      'estrae ciascun bit
       MID$(interruttore$, 16 - i%, 1) = "1"
    END IF
NEXT i%

'visualizza la lista degli equipaggiamenti
CLS : LOCATE 1, 1
PRINT "F E D C B A 9 8 7 6 5 4 3 2 1 0"
PRINT STRING$(32, CHR$(196))
FOR i% = 1 TO 16: PRINT MID$(interruttore$, i%, 1); " "; : NEXT
PRINT : PRINT STRING$(32, CHR$(196))
LOCATE 7, 1
PRINT "Lista equipaggiamenti ROM BIOS a 0040:0010"
PRINT
PRINT "0:   Drive floppy installati?"
PRINT "1:   Coprocessore matematico installato?"
PRINT "2-3: RAM originale della piastra madre del PC"
PRINT "4-5: Modalità video iniziale "
PRINT "6-7: Numero drive floppy"
PRINT "9-B: Numero porte seriali"
PRINT "C:   Adattatori joystick installati?"
PRINT "E-F: Numero stampanti parallele"
END

FUNCTION BitTest% (numero%, bit%)
   'estrae un bit dal numero
   BitTest% = -SGN(numero% AND 2 ^ bit%)
END FUNCTION

FUNCTION SInt% (num&)  'converte un intero lungo in un intero con segno
   SInt% = -((num& > 32767) * (num& - 65536)) - ((num& < 32767) * num&)
END FUNCTION
```

POKE

Si consulti anche

DEF SEG, PEEK, VARPTR

Scopo

L'istruzione POKE scrive un byte di dati in un indirizzo di memoria specificato nel segmento attivo; può essere utilizzata ad esempio per impostare lo stato del tasto CTRL.

Sintassi

```
POKE indirizzo, car
```

indirizzo è un'espressione di tipo intero nell'intervallo da 0 a 65535, che indica l'offset dell'indirizzo in cui POKE deve scrivere nel segmento di memoria in uso.

car è un valore intero nell'intervallo da 0 a 255, che deve essere scritto nell'indirizzo specificato.

Utilizzo

```
POKE &HC000, ASC("A")
```

Scrive il carattere A nell'offset C000H nel segmento attivo. Gli indirizzi possono essere specificati sia nella notazione esadecimale che in quella decimale.

Descrizione

L'istruzione POKE inserisce un singolo byte di dati (un valore nell'intervallo da 0 a 255) nell'indirizzo specificato. Il risultato di questa operazione dipende dall'indirizzo in cui viene scritto. Spesso POKE si limita a memorizzare il carattere in un'area della memoria non occupata, tuttavia alcuni indirizzi vengono utilizzati dal computer per determinati scopi, ovvero per fornire istruzioni all'hardware. Ad esempio, se si utilizza un monitor a colori il contenuto dello schermo viene memorizzato in memoria nel segmento B800H; scrivendo negli indirizzi di quell'area, i caratteri vengono visualizzati sullo schermo (l'indirizzo corrispondente per i computer con i monitor monocromatici è B000H).

L'istruzione DEF SEG può essere utilizzata per specificare il segmento di memoria in cui deve scrivere POKE. Se non viene definito un nuovo segmento, QBasic utilizza DGROUP, quello di default. Se si desidera leggere un valore diverso da un solo carattere, è necessario leggere il contenuto di due o più indirizzi di memoria vicini. Ad esempio, un intero viene memorizzato in 2 byte adiacenti, con il byte meno significativo (LSB) che precede quello più significativo (MSB); quest'ultimo indica il numero di volte in cui il valore intero può essere diviso per 256, mentre LSB è il resto di questa divisione. Il numero 1000, ad esempio, verrebbe memorizzato in memoria nel modo riportato nella Figura 19.3.

Quindi, per memorizzare un intero è necessario utilizzare POKE due volte, come nelle seguenti istruzioni:

```
POKE byte% ValoreInt% MOD 256
POKE byte% + 1, ValoreInt% \256
```

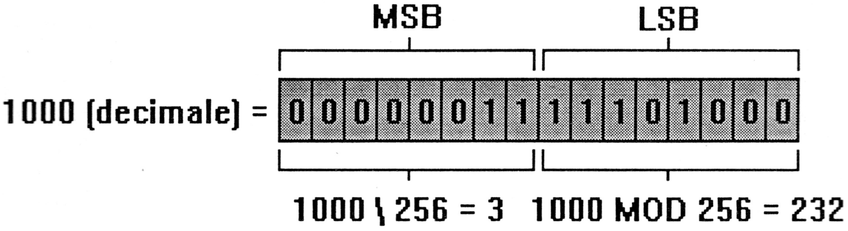

Figura 19.3 Il numero 1000 in memoria

Avvertenze

Quando si utilizza POKE è necessario prestare attenzione, in modo da non sovrascrivere nessun indirizzo utilizzato dal DOS o da altri programmi residenti, dato che, in caso contrario, il programma potrebbe interrompersi o il computer bloccarsi.

Esempio

Il programma di seguito riportato utilizza POKE per attivare il tasto BLOC MAIUSC. Il ROM BIOS registra lo stato dei tasti MAIUSC, CTRL, ALT, BLOC MAIUSC, BLOC NUM, BLOC SCORR e INS nella posizione 0040:0017 (segmento 40H, offset 17H) in memoria. Il programma passa innanzitutto al segmento dati del BIOS utilizzando DEF SEG e quindi attiva BLOC MAIUSC mediante POKE.

```
CLS
PRINT "Questo programma attiva il tasto BLOC MAIUSC per voi."

DEF SEG = &H40                'cambia il segmento dati del BIOS
POKE &H17, PEEK(&H17) OR 64   'attiva il tasto BLOC MAIUSC
DEF SEG                       'ristabilisce il DGROUP

INPUT "Immettere una riga di testo. Tutte le lettere saranno in maiuscolo: ",
testo$
```

WAIT

Si consulti anche

INP, OUT

Scopo

L'istruzione WAIT legge i dati dalla porta hardware specificata e sospende l'esecuzione del programma fino a quando non viene letta una determinata maschera di bit; può essere sfruttata per sospendere l'esecuzione, finché un dispositivo non è pronto a eseguire un particolare compito.

Sintassi

```
WAIT porta, espressione-and [,espressione-xor]
```

porta è l'indirizzo della porta da controllare e può essere un'espressione di tipo intero nell'intervallo da 0 a 65535.

espressione-and è un valore intero che viene utilizzato come operando in un'operazione logica AND con il byte letto dalla porta. È anche possibile utilizzare il valore intero espressione-xor come operando in un'operazione logica XOR (or esclusivo) con il byte che viene eseguito prima dell'operazione AND. L'esecuzione prosegue soltanto quando il risultato di queste operazioni è un valore diverso da 0.

Utilizzo

```
WAIT &H60, 127
```

Sospende l'esecuzione del programma e controlla la porta all'indirizzo 60H, fino a quando non viene impostato il bit superiore del byte letto.

Descrizione

L'istruzione WAIT risulta utile nei programmi che consentono di comunicare con altri computer. Ad esempio, se crea un programma che collega il proprio computer a un altro, è necessario controllare la porta hardware che collega i due computer; in caso contrario, si potrebbero perdere dei dati.

WAIT sospende l'esecuzione finché la porta di output specificata non restituisce un byte della maschera di bit indicata. Ogniqualvolta viene eseguita l'istruzione WAIT, QBasic legge un byte dalla porta specificata ed esegue un'operazione AND con l'espressione logica indicata nel secondo parametro. Ad esempio, se si desidera sospendere l'esecuzione del programma fino a quando non viene impostato il sesto bit del byte letto, è sufficiente specificare 32 come valore del secondo parametro: questo consente di mascherare tutti i bit, tranne il sesto. Se WAIT legge il valore 183 da una porta, l'operatore AND maschera il sesto bit nel seguente modo:

```
WAIT porta%, 32
```

```
    1 0 1 1 0 1 1 1 = 183
AND 0 0 1 0 0 0 0 0 = 32
-------------------------
    0 0 1 0 0 0 0 0 = 32
```

Dato che il risultato di questa operazione è un valore diverso da 0, l'esecuzione del programma riprende.

Per invertire la logica del test, ovvero per attendere fino a quando il sesto bit non viene impostato, è possibile eseguire un'operazione XOR preliminare con il byte letto, inserendo un terzo parametro nella chiamata a WAIT.

L'operazione XOR può essere utilizzata per attivare 1 o più bit, invertendo le impostazioni attuali. Anche in questo caso, si supponga che il byte letto dalla porta sia 183. Se si utilizza 32 come il valore per il secondo e il terzo parametro, l'istruzione esegue le seguenti operazioni:

WAIT porta%, 32, 32

```
    1 0 1 1 0 1 1 1 = 183       1 0 0 1 0 1 1 1 = 151
XOR 0 0 1 0 0 0 0 0 =32  AND    0 0 1 0 0 0 0 0 = 32
---------------------------------------------------
    1 0 0 1 0 1 1 1 =151        0 0 0 0 0 0 0 0 = 0
```

Si noti che l'istruzione WAIT non restituisce un valore. Se è necessario leggere le informazioni che la porta sta trasmettendo, occorre utilizzare la funzione INP.

Avvertenze

Se la maschera di bit specificata non si verifica in corrispondenza della porta in fase di controllo (ad esempio, se il computer remoto non riesce a rispondere al messaggio), WAIT rimane in un ciclo infinito. In questo caso, è necessario eseguire un reset.

Esempio

Il programma di seguito riportato utilizza WAIT per sospendere l'esecuzione fino a quando la stampante collegata alla prima porta parallela non è in linea. Dato che, quando la stampante è pronta, il sesto bit del byte all'indirizzo 3BDH è 0, per richiamare WAIT è sufficiente utilizzare il numero 32 nel secondo e nel terzo parametro.

```
CLS
PRINT "Assicurati che la stampante sia pronta."
WAIT &H3BD, 32, 32              'attende che la stampante parallela sia pronta
PRINT "Ora viene stampato un messaggio."
LPRINT "Guida di riferimento a QBasic", DATE$, TIME$, CHR$(12)
```

Capitolo 20

QBasic e il linguaggio macchina

INTRODUZIONE

Anche se QBasic è potente e dispone di diverse funzioni integrate, non è tuttavia adatto a eseguire determinate attività di programmazione. Il FORTRAN, ad esempio, è un linguaggio più indicato per applicazioni che devono eseguire calcoli scientifici e gestire continuamente numeri. Il C dispone di numerose funzioni atte alla gestione e allo spostamento di ampi blocchi di memoria, mentre i linguaggi assembler sono in grado di creare codici più rapidi ed efficaci per le applicazioni di basso livello e per il controllo dell'hardware del PC.

Un programma QBasic può rivolgersi alle risorse degli altri linguaggi, includendo routine in linguaggio macchina scritte con essi. Nella Tabella 20.1 sono riportate in ordine alfabetico le istruzioni e le funzioni di QBasic relative al linguaggio macchina.

CARICAMENTO DI ROUTINE IN LINGUAGGIO MACCHINA

QBasic fornisce un metodo che consente di incorporare routine non Basic nei propri programmi. Per caricare una routine in linguaggio macchina (ovvero una routine che è stata tradotta in un formato eseguibile) in una variabile QBasic e quindi eseguirla, è sufficiente utilizzare l'istruzione CALL ABSOLUTE.

Il codice in linguaggio macchina deve essere convertito in istruzioni DATA di QBasic, prima di poter essere caricato nella variabile; spesso viene caricato in un array di tipo intero.

Tabella 20.1 Le istruzioni e le funzioni relative al linguaggio macchina

Istruzione o funzione	Descrizione
CALL ABSOLUTE	Esegue una routine in linguaggio macchina che inizia a un indirizzo assoluto in memoria.
DEF SEG	Definisce il segmento attuale in memoria per le istruzioni e le funzioni che utilizzano gli indirizzi assoluti.
VARPTR, VARPTR$	Restituisce l'indirizzo dell'offset di una variabile nel segmento di dati in uso.
VARSEG	Restituisce l'indirizzo del segmento di una variabile.

Per utilizzare l'istruzione CALL ABSOLUTE, è necessario seguire la seguente sintassi:

```
CALL ABSOLUTE(indirizzo)
```

indirizzo è un intero che indica l'indirizzo in memoria della variabile che contiene la routine in linguaggio macchina. QBasic fornisce la funzione VARPTR, che restituisce l'indirizzo di una variabile numerica, di un array o di una stringa a lunghezza fissa. L'indirizzo restituito da questa funzione rappresenta l'offset della variabile specificata nel relativo segmento di memoria.

L'indirizzo restituito da VARPTR può essere sfruttato per il primo elemento dell'array, ovvero è sufficiente utilizzare la seguente istruzione:

```
indirizzo% = VARPTR(codiceArray%(0))
```

Tuttavia, dato che QBasic non memorizza gli array dinamici in DGROUP, è indispensabile passare al segmento di memoria corretto, prima di richiamare la routine. La funzione VARSEG restituisce l'indirizzo del segmento della variabile, mentre l'istruzione DEF SEG consente di impostarlo come segmento attivo. Per eseguire una routine in linguaggio macchina caricata in un array dinamico, è sufficiente utilizzare le seguenti istruzioni:

```
DEF SEG = VARSEG(codiceArray%(0))
CALL ABSOLUTE(VARPTR(codiceArray%(0)))
```

Dato che ogni elemento di un array intero può contenere 2 byte di codice macchina, quando l'array viene dimensionato è necessario utilizzare soltanto metà del numero degli indici, come il numero di byte della routine. L'istruzione POKE può essere sfruttata per caricare il codice direttamente nella memoria occupata dall'array, come dimostrato nel seguente spezzone di programma:

```
DEF SEG = VARSEG(codiceArray%(0))        'imposta il segmento corretto
offset% = VARPTR(codiceArray%(0))        'trova l'inizio dell'array

FOR i% = 0 TO 30
    READ byte%
    POKE offset% + i%, VAL("&H" + byte%)
NEXT i%
```

```
DATA  55,8B,EC,06,1E,57,56,2E,C7,06
.
.
.
```

È necessario prestare attenzione quando si carica un array e si richiama una routine; in particolare, è indispensabile assicurarsi che QBasic non sposti la variabile, prima che la routine venga richiamata. Per impedire che questo avvenga, è sufficiente utilizzare gli array dinamici per memorizzare la routine; dato che QBasic li inserisce nei relativi segmenti all'esterno di DGROUP, essi non possono essere spostati.

Conversione del linguaggio macchina in istruzioni DATA

Per utilizzare le routine in linguaggio macchina con l'istruzione CALL ABSOLUTE, è necessario convertirle in istruzioni DATA, che possono essere utilizzate nei programmi QBasic. A questo scopo, uno dei metodi è quello di caricare la routine in un programma di debug, ad esempio Debug o Microsoft CodeView, copiando quindi i valori in byte dallo schermo in istruzioni DATA del proprio programma. Tuttavia, il programma di seguito riportato fornisce un metodo migliore per effettuare la conversione, leggendo un file che contiene codice in linguaggio macchina e traducendo ogni byte in istruzioni DATA di QBasic, che possono quindi essere integrate nel proprio programma. Il programma calcola inoltre un valore di somma totale che la routine in fase di caricamento può utilizzare per individuare i dati immessi in modo errato e quindi inserisce il valore stesso nell'ultima istruzione DATA.

```
'BIN2DATA.BAS - traduce file binari in istruzioni DATA
BinFile$ = "ROUTINE.COM"        'cambia il nome del file in uno
                                'tipico dei file in linguaggio macchina
dataFile$ = "HEXDATA.INC"

OPEN BinFile$ FOR BINARY AS #1 LEN = 1
OPEN dataFile$ FOR OUTPUT AS #2

DIM byte AS STRING * 1

dataRiga$ = "DATA "
cont% = 0: controllaSomma& = 0

DO UNTIL EOF(1)
   GET #1, , byte
   controllaSomma& = controllaSomma& + ASC(byte)
   dataRiga$ = dataRiga$ + RIGHT$("0" + HEX$(ASC(byte)), 2)
   cont% = cont% + 1
   IF cont% < 15 THEN
      dataRiga$ = dataRiga$ + ", "
   ELSE
      PRINT #2, dataRiga$
      dataRiga$ = "DATA "
      cont% = 0
   END IF
LOOP

IF cont% > 0 THEN
   dataRiga$ = LEFT$(dataRiga$, LEN(dataRiga$) - 2)
   PRINT #2, dataRiga$
END IF
```

```
dataRiga$ = "DATA " + STR$(controllaSomma&)
PRINT #2, dataRiga$
CLOSE
END
```

Bibliografia

Craigh, John Clark. *Microsoft QuickBASIC Programmer's Toolbox.* Redmond, Wash.; Microsoft Press, 1988.

Duncan, Ray. *Advanced MS-DOS Programming.* 2d ed. Redmond, Wash.; Microsoft Press, 1988.

Duncan, Ray. *IBM ROM BIOS: Programmer's Quick Reference.* Redmond, Wash.; Microsoft Press, 1988.

Norton, Peter and Richard Wilton. *The New Peter Norton Programmer's Guide to the IBM PC* & *PS/2.* 2d ed. Redmond, Wash.; Microsoft Press, 1988.

The Waite Group. *MS-DOS Developer's Guide.* Indianapolis, Ind.: Howard W. Sams & Co., 1989

The Waite Group. *MS-DOS Papers.* Indianapolis, Ind.: Howard W. Sams & Co., 1988.

CALL ABSOLUTE

Si consulti anche

CALL, DEF SEG, VARPTR, VARSEG

Scopo

L'istruzione CALL ABSOLUTE esegue una routine in linguaggio macchina che è stata caricata in una variabile in corrispondenza di un indirizzo assoluto in memoria; consente di utilizzare le routine sviluppate con versioni precedenti di Basic.

Sintassi

```
CALL ABSOLUTE([var[,var...]]indirizzo)
```

var indica una o più variabili facoltative fornite come parametri alla routine in linguaggio macchina.

indirizzo è un'espressione valida di QBasic o un nome di variabile che esprime numericamente un valore intero, che rappresenta l'offset della routine richiamata nel segmento attualmente definito.

Utilizzo

```
CALL ABSOLUTE(&HF000)
```

Trasferisce il controllo alla routine in linguaggio macchina in memoria all'offset 61440 (F000H) nel segmento attivo.

Descrizione

La maggior parte dei programmatori che passa a QBasic dopo aver utilizzato versioni precedenti di Basic, dispone già di numerose routine in linguaggio macchina. Queste consentono di migliorare la visualizzazione, di accelerare l'esecuzione, talvolta lenta, di sezioni di codice oppure di eseguire un'operazione di difficile esecuzione in Basic. L'istruzione CALL ABSOLUTE consente di trasportare nei programmi QBasic le routine in linguaggio macchina sviluppate in Basic.

Dato che ogni array dinamico viene memorizzato nel proprio segmento nel far heap, se il programma creato utilizza una routine in linguaggio macchina memorizzata in un array dinamico, è sufficiente sfruttare la funzione VARSEG per individuare il segmento dell'array, utilizzando quindi l'istruzione DEF SEG per modificare il segmento attuale.

CALL ABSOLUTE fornisce un alto grado di compatibilità con le versioni precedenti di Basic, che spesso utilizzano l'istruzione BLOAD per caricare nell'indirizzo assoluto in memoria i file in linguaggio macchina.

Esempio

Gli errori di esecuzione in QBasic spesso si verificano quando un programma cerca di inviare l'output a una stampante priva di carta o spenta. Il programma di seguito riportato utilizza una routine in linguaggio macchina, che viene memorizzata in un array e utilizzata con CALL ABSOLUTE, per verificare lo stato della stampante, prima dell'output; se la stampante non è disponibile, vengono intraprese le azioni corrette.

```
'$DYNAMIC                  'rende dinamici tutti gli array

DIM mCode(0 TO 11) AS INTEGER             'dimensiona l'array per la routine
Stampante% = 1                            '1=LPT1:,2=LPT2: ecc.
testSomma& = 0                            'impostazione per il test di
controllo-somma
DEF SEG = VARSEG(mCode(0))                'imposta l'array del segmento
offset% = VARPTR(mCode(0))                'ottiene l'indirizzo dell'array
RESTORE Codice

FOR i% = 0 TO 23                          'legge ciascun byte di codice
macchina
    READ byte%
    testSomma& = testSomma& + byte%
    POKE offset% + i%, byte%              'legge i byte nell'array
NEXT i%

READ controllaSomma&
IF testSomma& <> controllaSomma& THEN     'controlla per errori di battitura
   PRINT "Errore di controllo somma!!!"
   BEEP: PRINT "Non puoi continuare"
   STOP
END IF

CALL absolute(Stampante%, offset%)        'esegue la routine
DEF SEG                                   'ristabilisce il segmento di default

IF Stampante% = 144 OR Stampante% = 208 THEN 'indica se la stampante è pronta
   PRINT "La stampante è pronta per l'output."
ELSE
   PRINT "La stampante non è pronta per l'output."
END IF
END
```

```
'byte di dati per la routine in linguaggio macchina e il controllo somma
Codice:
   DATA &H55,&H8B,&HEC,&H8B,&H5E,&H06,&H8B,&H17
   DATA &H4A,&HB4,&H02,&HCD,&H17,&H8A,&HC4,&HB4
   DATA &H00,&H89,&H07,&H5D,&HCA,&H02,&H00,&H00
   DATA 2300
```

DEF SEG

Si consulti anche

BLOAD, BSAVE, CALL ABSOLUTE, PEEK, POKE, VARPTR, VARPTR$, VARSEG

Scopo

L'istruzione DEF SEG indica il segmento utilizzabile per le istruzioni e le funzioni di QBasic relative a indirizzi in memoria. L'architettura in chip del PC suddivide la memoria in aree di 64 KB, ognuna definita dall'indirizzo di partenza, ovvero dal numero di segmento. Questa istruzione può essere sfruttata per impostare l'indirizzo del segmento attuale, in modo che possa essere utilizzato da una successiva funzione PEEK o dalle istruzioni BLOAD, BSAVE, CALL ABSOLUTE, POKE.

Sintassi

```
DEF SEG [=segmento]
```

segmento è una qualsiasi costante, variabile o espressione che esprime numericamente un valore intero nell'intervallo da 0 a 65535. Se questo parametro viene omesso, QBasic utilizza il segmento di dati di default, ovvero DGROUP.

Dato che il massimo intero decimale con segno è 32767, per impostare il valore del segmento spesso si utilizzano numeri esadecimali.

Utilizzo

```
DEF SEG = &HB800
```

Imposta l'indirizzo del segmento attuale nel buffer video per i sistemi che utilizzano un adattatore CGA.

```
DEF SEG
```

Assegna nuovamente il segmento attuale al segmento dati di default di QBasic, ovvero DGROUP.

Descrizione

Diverse istruzioni e funzioni di QBasic consentono di fare riferimento ai dati memorizzati in un indirizzo assoluto nella memoria del computer. La funzione PEEK, ad esempio, restituisce il contenuto del byte nella posizione indicata; l'istruzione CALL ABSOLUTE esegue un programma in linguaggio macchina, memorizzato nell'indirizzo specificato. Queste istruzioni e funzioni richiedono che l'indirizzo sia un intero nell'intervallo da 0 a 65535, ovvero il valore che indica una posizione in un blocco di 64 KB di memoria.

Per accedere a un segmento nei 1024 KB di memoria indirizzabile, è necessario innanzitutto specificare l'indirizzo di partenza, definito indirizzo del segmento, di un blocco di 64 KB. Ad esempio, potrebbe essere necessario accedere alla memoria esterna a DGROUP, per caricare (utilizzando BLOAD) un file grafico direttamente nel buffer video; DEF SEG consente di indicare il segmento di 64 KB in cui si verifica questa operazione. A meno che non venga immessa un'altra istruzione DEF SEG, QBasic interpreta qualsiasi indirizzo specificato nella funzione PEEK o nelle istruzioni BLOAD, BSAVE, CALL ABSOLUTE o POKE come offset di questo segmento. Se viene specificato un segmento diverso, il programma non si interrompe, a meno che la memoria in fase di utilizzo da parte del sistema operativo non venga sovrascritta da QBasic o dal programma stesso.

Errori

Se viene specificato un indirizzo del segmento al di fuori dell'intervallo da 0 a 65535, QBasic visualizza il messaggio di errore: "Chiamata di funzione non lecita" e non modifica il segmento attuale.

Esempio

Per meglio comprendere l'istruzione DEF SEG, si esamini l'utilizzo dell'area dati del BIOS nella memoria bassa, partendo dal segmento 9940H. Quest'area contiene il buffer della tastiera, le informazioni relative allo stato dell'apparecchiatura installata e quelle relative al video.

Il programma di seguito riportato utilizza le informazioni presenti nell'area dati del BIOS per scrivere del testo da QBasic direttamente nella riga inferiore dello schermo, senza per questo farlo scorrere. Viene innanzitutto controllata l'area dati del BIOS per verificare se il programma è in fase di esecuzione su un computer con un video monocromatico o con video a colori. Questo consente di determinare a quale segmento di memoria inviare l'output. Per eseguire questo controllo, il programma sfrutta DEF SEG per impostare il segmento attuale nell'area dati del BIOS, che inizia a 40H.

Il programma verifica inoltre le dimensioni dello schermo, definite nel segmento dati del BIOS, per determinare se la visualizzazione è in modalità 40 colonne o 80 e se il numero di righe è 25, 43 o 50. Queste informazioni consentono di calcolare l'indirizzo dell'offset (nella memoria video) della riga inferiore dello schermo.

Infine, il programma invia il testo, utilizzando nuovamente DEF SEG per impostare il segmento corretto. Si noti che ogni carattere dell'output richiede 2 byte di memoria video, uno per il codice ASCII del carattere e l'altro per il relativo attributo di visualizzazione o colore.

```
DEF SEG = &H40              'imposta il segmento per l'area dati del BIOS
IF PEEK(&H49) = 7 THEN  'controlla la modalità video
   segmentoVideo% = &HB000       'la modalità 7 indica un video monocromatico
ELSE
   segmentoVideo% = &HB800       'ogni altra modalità indica un video a colori
END IF

colonne% = PEEK(&H4A) + (PEEK(&H4B) * 256) 'ottiene la larghezza dello schermo
byte% = PEEK(&H4C) + (PEEK(&H4D) * 256)    'ottiene le dimensioni del video
righe% = byte% \ (colonne% * 2)            'calcola il numero di righe
```

```
offset% = (colonne% * (righe% - 1)) * 2     'ottiene l'ultima riga

DEF SEG = segmentoVideo%          'imposta il segmento alla memoria video
FOR i% = 0 TO colonne% - 1
    POKE offset% + (i% * 2), ASC("*")       'scrive un carattere
    POKE offset% + (i% * 2) + 1, 78         'scrive un attributo
NEXT i%
DEF SEG                                     'ristabilisce il segmento DGROUP
SLEEP 3
END
```

VARPTR, VARPTR$

Si consulti anche

DEF SEG, VARSEG

Scopo

La funzione VARPTR restituisce l'indirizzo di offset di una variabile all'interno del relativo segmento di memoria. Se è necessario caricare in memoria una routine in linguaggio macchina, è possibile ad esempio utilizzare VARPTR per individuare l'indirizzo della variabile in cui queste istruzioni possono essere caricate. La funzione VARTPR$ restituisce una stringa contenente l'indirizzo della variabile; può essere utilizzata nelle chiamate a DRAW o a PLAY per eseguire le sottostringhe contenute in una variabile.

Sintassi

```
VARPTR(var)
VARPTR$(var)
```

var è una qualsiasi variabile corretta di QBasic. Se la variabile indicata non esiste, QBasic la crea prima di restituire l'indirizzo o la stringa contenente l'indirizzo.

Utilizzo

```
offSet% = VARPTR(codiceArray%(0))
```

Assegna alla variabile offset% l'indirizzo dell'offset del primo elemento nell'array intero codiceArray%.

```
PLAY "X" + VARTPR$(cmd$)
```

Suona la musica presente nei comandi contenuti nella variabile stringa cmd$.

Descrizione

Qualsiasi riferimento a una posizione in memoria consiste di due parti: il segmento, un valore intero (nell'intervallo da 0 a 65535), che indica il blocco di memoria di 64 KB in cui risiede l'oggetto e l'offset, un altro valore intero (nell'intervallo da 0 a 65535), che indica la posizione dell'oggetto all'interno del segmento di 64 KB. La funzione VARSEG restituisce la parte di segmento di un indirizzo di variabile, mentre la funzione VARPTR restituisce l'offset dell'indirizzo sotto forma di stringa.

Dato che QBasic memorizza le variabili semplici e le stringhe in DGROUP, è sufficiente utilizzare VARPTR per indirizzare queste variabili. Dato che gli array numerici e quelli dinamici di stringhe a lunghezza fissa vengono invece memorizzati in segmenti propri, all'esterno di DGROUP, è necessario utilizzare sia VARSEG che VARPTR per indirizzare questi segmenti.

Si noti che, quando si utilizza una di queste due istruzioni per ottenere l'indirizzo di un array, è necessario specificare un numero dell'elemento e il nome dell'array. Dato che i dati dell'array vengono memorizzati in modo continuo, l'indirizzo del primo elemento di un array, è in effetti l'indirizzo di tutto l'array.

Quando VARPTR viene utilizzata con una stringa a lunghezza fissa, non restituisce l'indirizzo di quest'ultima, ma un puntatore a un descrittore di stringa di 4 byte presente in DGROUP. I primi 2 byte del descrittore contengono la lunghezza della stringa, gli altri 2 byte invece contengono l'indirizzo della stringa, che è l'offset dall'inizio di DGROUP.

Per semplificare i compiti di programmazione è necessario utilizzare sempre sia VARSEG che VARPTR, quando deve essere specificato l'indirizzo di un array numerico.

Si noti che QBasic memorizza sempre gli array delle variabili a lunghezza fissa in DGROUP.

Avvertenze

QBasic alloca in modo dinamico lo spazio di memorizzazione in DGROUP. Se viene creata una nuova variabile, oppure ne viene utilizzata una all'interno di una funzione o di un'espressione, QBasic potrebbe cambiarne l'indirizzo. Per risolvere questo problema, il programma deve sempre utilizzare l'indirizzo restituito immediatamente da VARPTR; in caso contrario, QBasic può muovere la variabile, rendendo errato l'indirizzo restituito.

Esempio

QBasic non dispone di una funzione che consente di individuare l'esistenza o meno di un file. Se si cerca di aprire un file non esistente, QBasic lo crea utilizzando una lunghezza di 0 byte oppure, se il file è stato aperto soltanto per operazioni di input, visualizza un messaggio di errore. Queste condizioni di errore possono essere rilevate, ma il codice di individuazione è spesso mal costruito.

I servizi del DOS comprendono diverse funzioni che restituiscono informazioni relative ai file specificati. Il programma di seguito riportato utilizza una routine in linguaggio macchina (che può essere incorporata in altri programmi) la quale richiede al DOS le dimensioni (in byte) di un determinato file e accetta un nome contenente i caratteri jolly (* e ?). Se questo nome viene fornito, il programma restituisce la somma delle dimensioni di tutti i file corrispondenti; se la routine restituisce 0, ciò significa che il file non esiste.

Il programma utilizza una routine in linguaggio assembler memorizzata in una variabile array per maggiore compatezza.

Vengono lasciati vuoti i primi 29 elementi dell'array che contiene il codice in linguaggio macchina; questi elementi formano un indirizzo di trasferimento temporaneo su disco e il buffer del nome del file. Il codice effettivo della routine inizia in corrispondenza dell'elemento 30, mentre il valore restituito dalla routine viene

assegnato all'elemento 1.

```
'$DYNAMIC                  'rende dinamici tutti gli array

DIM mCode(1 TO 70) AS LONG     'dimensiona l'array per la routine in
                               'linguaggio macchina
DIM Byte AS STRING * 2         'mantiene ciascuna voce dei dati

DEF SEG = VARSEG(mCode(1))     'imposta l'array del segmento
offset% = VARPTR(mCode(30))    'ottiene l'indirizzo dell'array numero 30

'legge i byte di dati in linguaggio macchina nell'array
RESTORE
FOR i% = 0 TO 161
    READ Byte
    POKE offset% + i%, VAL("&H" + Byte)
NEXT i%

CLS
PRINT "Questo programma riporta le dimensioni del file specificato."
PRINT "------------------------------------------------------------"
PRINT "Il nome file può includere il percorso e può"
PRINT "utilizzare i caratteri sostitutivi ? e *."
PRINT "Il programma restituisce la dimensione del file in byte,"
PRINT "o, se trova più file con lo stesso nome, la somma"
PRINT "delle dimensioni di tutti i file. Se la dimensione restituita è 0,"
PRINT "il file non esiste (o almeno non è nella directory specificata)."
PRINT
PRINT "Immettere il percorso del file (fino a 64 caratteri),"
PRINT "o premere Invio per terminare."

DO
    LOCATE 14, 1: PRINT SPACE$(240);
    LOCATE 14, 1: INPUT "Percorso"; percorso$
    IF LEN(percorso$) > 1 AND LEN(percorso$) < 65 THEN
       CALL absolute(percorso$, VARPTR(mCode(30)))      'chiama la routine
       LOCATE 16, 1: PRINT "Dimensioni = "; mCode(1) 'visualizza le dimensioni
       LOCATE 20, 1, 1: PRINT "Premi qualsiasi tasto per continuare. ";
       r$ = INPUT$(1)
       LOCATE 20, 1: PRINT SPACE$(26);
    END IF
LOOP UNTIL percorso$ = ""
DEF SEG          'ristabilisce il segmento
END

'routine in linguaggio macchina
    DATA 55,8B,EC,06,1E,57,56,2E,C7,06,00,00,00,00,2E
    DATA C7,06,02,00,00,00,0E,07,8B,5E,06,8B,0F,83,F9
    DATA 01,72,6F,83,F9,40,77,6A,8B,77,02,BF,33,00,FC
    DATA F3,A4,B0,00,AA,0E,1F,B4,2F,CD,21,2E,89,1E,2F
    DATA 00,2E,8C,06,31,00,BA,04,00,B4,1A,CD,21,BA,33
    DATA 00,33,C9,B4,4E,CD,21,72,2D,2E,A1,1E,00,2E,8B
    DATA 16,20,00,2E,A3,00,00,2E,89,16,02,00,B4,4F,CD
    DATA 21,72,15,2E,A1,1E,00,2E,8B,16,20,00,2E,01,06
    DATA 00,00,2E,11,16,02,00,EB,E5,2E,A1,31,00,2E,8B
    DATA 16,2F,00,8E,D8,B4,1A,CD,21,2E,8B,16,02,00,2E
    DATA A1,00,00,5E,5F,1F,07,5D,CA,02,00,00
```

VARSEG

Si consulti anche

DEF SEF, VARPTR, VARTPR$

Scopo

La funzione VARSEG restituisce la parte di segmento di un indirizzo di variabile; può ad esempio essere sfruttata per individuare il segmento di un'array numerico dinamico, che si desidera salvare su disco utilizzando BSAVE.

Sintassi

```
VARSEG(var)
```

var è un qualsiasi nome di variabile corretto di QBasic. Se la variabile indicata non esiste, QBasic la crea prima di restituire l'indirizzo.

Utilizzo

```
segmento% = VARSEG(codiceArray%(0))
```

Assegna alla variabile segmento% il segmento del primo elemento contenuto nell'array di tipo intero codiceArray%.

Descrizione

VARSEG restituisce il segmento occupato da qualsiasi variabile, consentendo in questo modo di specificarne l'indirizzo. Questa funzione restituisce un numero intero che corrisponde all'indirizzo del segmento contenente la variabile specificata; se quest'ultima è una variabile numerica semplice, oppure a lunghezza fissa, il valore restituito indica l'indirizzo del segmento DGROUP. Dato che DGROUP è il segmento di default di QBasic, non è necessario specificare la parte di segmento dell'indirizzo, quando si fa riferimento ai dati memorizzati in quella posizione; di solito è sufficiente indicare l'offset dall'inizio del segmento (che viene restituito dalla funzione VARPTR).

Se la variabile è un array numerico dinamico, oppure un array dinamico di stringhe a lunghezza fissa, VARSEG restituisce l'indirizzo del segmento che QBasic alloca a essa. I programmi devono indicare sia il segmento (utilizzando VARSEG), che l'offset (mediante VARPTR) di questa variabile, quando ne passano l'indirizzo.

Esempio

Si consulti la voce VARPTR per esaminare un programma che utilizza sia VARSEG che VARPTR.

Capitolo 21

Debug

UNA SESSIONE DI DEBUG

In questa parte viene illustrata, mediante un programma di esempio, una sessione di debug interattiva. Si immagini di aver creato il programma ORDFILE.BAS, riportato di seguito, che ha lo scopo di ordinare un file a lunghezza fissa in memoria. Tuttavia, ORDFILE.BAS contiene diversi errori, che ne impediscono un'esecuzione corretta; per individuarli e correggerli, è possibile utilizzare l'ambiente di QBasic.

```
'ORDFILE.BAS -ordina un file a lunghezza fissa
DECLARE FUNCTION OrdinaFile% (percorso$, recLungh%, offset%, fieldLend%,
ordinaErr%)

CONST FALSO = 0, VERO = NOT FALSO

TYPE Elemento
     sChiave AS STRING * 10
     recNum AS INTEGER
END TYPE

PRINT : PRINT "ORDFILE - ordina file ad accesso casuale": PRINT

'acquisisce informazioni per l'ordinamento del file
LINE INPUT "Immettere il percorso del file da ordinare ?"; percorso$
LINE INPUT "Qual è la lunghezza del file da ordinare ? "; temp$
recLungh% = VAL(temp$)
LINE INPUT "Definire il campo d'ordinamento - posizione iniziale ? "; temp$
offset% = VAL(temp$)
LINE INPUT "                       -lunghezza del file ? "; temp$
campoLungh% = VAL(temp$)

IF OrdinaFile%(percorso$, recLungh%, offset%, campoLungh%, ordinaErr%) THEN
   PRINT "File ordinato con successo"
ELSE
   PRINT "Impossibile ordinare il file": BEEP
END IF
END
```

```
'ordina il file in memoria

FUNCTION OrdinaFile% (percorso$, recLungh%, offset%, campoLungh%, ordinaErr%)
STATIC
   ordinaErr% = FALSO


   'trova la lunghezza del file
   OPEN percorso$ FOR RANDOM AS #1 LEN = recLungh%
   fileLungh& = LOF(1)
   CLOSE 1

   'controlla che l'ordinamento sia possibile
   record& = fileLungh \ recLungh%          '(prima definizione di record&)
   IF (campoLungh% + offset%) > recLungh% THEN
      ordinaErr% = 1: EXIT FUNCTION

   END IF

   'crea un file temporaneo
   tempFile$ = UCASE$(percorso$)
   punPosizione% = INSTR(tempFile$, ".")
   IF punPosizione% > 0 THEN
      tempFile$ = LEFT$(tempFile$, punPosizione%) + "TMP"
   ELSE
      tempFile$ = tempFile$ + ".TMP"
   END IF
   cmd$ = "COPY " + percorso$ + " " + tempFile$ + " > nul"
   SHELL cmd$

   'prepara l'ordinamento
   DIM array(1 TO record%)  AS Elemento

  OPEN percorso$ FOR RANDOM AS #1 LEN = recLungh%
  IF offset% > 1 THEN                'definisce i record per leggere il file
     FIELD 1, offset% - 1 AS fittizio$, campoLungh% AS Chiavel$
  ELSE
     FIELD 1, campoLungh% AS Chiavel$
  END IF
  FIELD 1, recLungh% AS finaleRec$
  i% = 1


  'riempie l'array con le chiavi d'ordinamento dal file
  DO
     GET 1, i%
     LSET array(i%).sChiave = Chiavel$      'inserisce ciascuna chiave
                                            'd'ordinamento nell'array
     array(i%).recNum = i%     'inserisce ciascun numero record nell'array
     i% = i% + 1
  LOOP UNTIL i% > record&

  'realizza l'ordinamento
  partizione% = record% \ 2
  DO WHILE partizione% > 0
     gamma% = record& - partizione%
     DO
         scambio% = FALSO
         FOR i% = 1 TO gamma%  'se necessario effettua lo scambio degli
                               'elementi dell'array
              IF array(i%).sChiave > array(i% + partizione%).sChiave THEN
              SWAP array(i%), array(i% + partizione%)
              scambio% = i%
        END IF
     NEXT i%
```

```
      gamma% = scambio%
    LOOP UNTIL NOT scambio%
    partizione% = partizione% \ 2
  LOOP

  'utilizza l'array ordinato per ordinare il file
  OPEN tempFile$ FOR RANDOM AS #2 LEN = recLungh% 'apre un file temporaneo
  FIELD 2, recLungh% AS tempRec$                  'definisce i record per il file
  FOR i% = 1 TO record&
      GET 2, array(i%).recNum               'ottiene un record dal file
      tempRecVar$ = tempRec$     'assegna un record ad una variabile temporanea
      LSET finaleRec$ = tempRecVar$ 'assegna una variabile temporanea a un campo
      PUT 1, i%                             'scrive il campo nel file
  NEXT i%


      CLOSE : KILL tempFile$                'cancella il file temporaneo
      OrdinaFile% = VERO
  END FUNCTION
```

Il listato di seguito riportato illustra il file ESEMPIO.DAT, che contiene i dati non ordinati utilizzati per sottoporre a test il programma.

```
89/03/01 1549 C00003 ICIPA       1000      2.50 B000001
89/03/01 1645 C00008 AAC          500      5.80 B000002    S000030
89/03/01 1646 C00013 CWC          250      5.00 B000003    S000044
89/03/01 1648 C00011 ALMO         500     10.00 S000004    B000006
89/03/01 1649 C00002 QCLGA        180      1.92 B000005    S000017
89/03/01 1651 C00005 ALMO         500     10.00 B000006    S000004
89/03/02 1648 C00011 ALMO        1000     10.00 S000004A   B000009
89/03/02 1701 C00007 WPL          200      3.65 S000007    B000072
89/03/01 1703 C00012 CIG          700      5.80 S000008
89/03/02 1704 C00006 ALMO        1000     10.00 B000009    S000004A
89/03/02 1075 C00003 ASD          300     17.00 S000010
89/03/03 1706 C00009 ALMO         500     10.00 B000011    S000004
89/03/03 2042 C00002 POSO         100      6.00 S000014    B000073
89/03/04 2044 C00012 BUD          500      3.00 B000015
89/03/04 2045 C00013 ALMO         300     10.00 B000016    S000004C
89/03/04 2046 C00014 QCLGA        180      1.92 S000017    B000005
89/03/04 2047 C00007 BAB          300       .75 S000018
89/03/04 2049 C00001 ALMO         500     10.00 B000019    S000004D
89/03/05 2050 C00011 CCYOA       1000      2.42 B000020    A000023
89/03/05 2051 C00020 WOIPA       1000      6.65 S000021
89/03/05 2052 C00019 WIM         5000      0.10 B000022
89/03/06 2053 C00021 CCYDA       1000      2.24 S000023    B000020
89/03/07 2053 C00021 CCYOA       2000      2.24 S000023A   B000026
89/03/07 1648 C00011 ALMO         500     10.00 S000004D   B000019
89/03/08 1648 C00011 ALMO        2000     10.00 S000004E   B000028
89/03/08 2125 C00018 WPL         2000      3.75 S000024
89/03/10 2126 C00025 ERA         8000       .05 S000025
89/03/11 2128 C00026 CCYOA       2000      2.24 B000026    S000023A
89/03/11 2129 C00014 AUP          100      5.18 B000027    S000041
89/03/11 2130 C00015 ALMO        2000     10.00 B000028    S000004E
89/03/12 2131 C00015 WLW          400      3.69 B000029
89/03/12 2132 C00018 AAC          500      5.80 S000030    B000002
```

Prima di iniziare, è bene fare una copia di lavoro di questo file di dati (assegnandogli il nome ESEMPIO.TST).

A questo punto, è necessario caricare il programma ORDFILE.BAS nell'ambiente QBasic. Il modo migliore per sottoporre a test il programma consiste nell'eseguirlo, premendo MAIUSC+F5 oppure selezionando Avvia dal menu Esegui.

Viene visualizzata una richiesta nella finestra Output. Il programma sembra aver eseguito correttamente l'operazione di compilazione, dato che l'editor di QBasic sarebbe stato in grado di rintracciare un eventuale errore di sintassi, prima di giungere a questo punto. Ora vengono richiesti dei dati. In risposta al primo messaggio: "Immettere il percorso del file da ordinare", è sufficiente immettere **ESEMPIO.TST**. Quindi, come risposta alla richiesta successiva, è necessario immettere **64**, la lunghezza di ogni record nel file ESEMPIO.TST. Adesso, il programma richiede le informazioni relative al campo di ordinamento.

Il programma è stato progettato in modo da poter utilizzare come chiave di ordinamento qualsiasi campo del record del file. Si utilizzi il campo del codice del prodotto in ESEMPIO.TST, che inizia alla posizione 22 nel record e ha una lunghezza di cinque caratteri. Dopo aver risposto alle richieste, lo schermo contiene le seguenti informazioni:

```
Immettere il percorso del file da ordinare              ? ESEMPIO.TST
Qual è la lunghezza del record del file                 ? 64
Definisce il campo d'ordinamento - posizione iniziale  ? 22
                                 - lunghezza del campo ? 5
```

Il programma genera immediatamente un errore. QBasic visualizza il messaggio: "Indice fuori limite" in una finestra di dialogo e posiziona il cursore in corrispondenza della riga che ha provocato l'errore. Questa finestra di dialogo indica all'utente che, quando il programma ha cercato di dimensionare l'array, un indice era fuori limite.

Qual è la ragione di questo messaggio di errore, quando in effetti l'array viene dimensionato solo ora? Per rispondere a questa domanda, è necessario esaminare l'array. Questo è un array dinamico, dato che si utilizza una variabile per dimensionarlo. Ora, si esamini il contenuto della variabile record%; è necessario premere ESC o INVIO per eliminare la finestra di dialogo dallo schermo.

Il metodo più semplice per eseguire questa operazione in QBasic consiste nell'utilizzare la finestra Immediato. Quando la finestra di dialogo viene eliminata, il cursore è posizionato sulla riga che ha provocato l'errore; è necessario spostarlo all'inizio della variabile record%, premendo quindi MAIUSC+CTRL+→ per selezionarla e copiarla poi nella clipboard premendo CTRL+INS oppure selezionando il comando Copia dal menu Modifica.

La finestra Immediato consente di immettere le istruzioni QBasic da eseguire, senza che diventino parti permanenti del proprio programma. Questa caratteristica risulta particolarmente utile soprattutto nel debug, dato che consente di utilizzare l'istruzione PRINT per visualizzare i valori delle variabili da controllare, in questo caso record%. Premendo F6, è possibile passare dalla finestra Edit a quella Immediato. Ora, è necessario immettere l'istruzione PRINT seguita da uno spazio, oppure semplicemente il punto di domanda, che ne rappresenta l'abbreviazione. Quindi, dopo aver copiato il contenuto della clipboard (premendo CTRL+INS) nella finestra Immediato, è sufficiente premere INVIO perché QBasic visualizzi il valore di record% nella finestra Output. Dato che questo valore viene visualizzato nell'ultima posizione del cursore, talvolta risulta difficile localizzare l'output. La finestra Immediato illustra la causa dell'errore. Infatti, la variabile record%, che dovrebbe contenere il numero dei record nei file, contiene invece il valore 0. A questo punto è necessario prendere in esame la parte di codice che ha impostato il valore di record%; per passare nuovamente alla finestra Edit è sufficiente premere un tasto qualsiasi per ritornare alla finestra Immediato e quindi premere F6.

Esaminando le righe presenti sopra quella evidenziata, fino a quella che contiene il commento (prima definizione di record&), è possibile notare che vi è un errore. La variabile che contiene il numero dei record è in effetti denominata record& (una variabile di tipo intero lungo) e non record% (una variabile di tipo intero corto). Dato che con la variabile utilizzata per dimensionare l'array è stato inserito un carattere di dichiarazione del tipo intero, QBasic è partito dal presupposto che si stava facendo riferimento a una nuova variabile e ha quindi creato record%. Questa è la ragione del valore 0.

Per risolvere il problema, è necessario posizionare il cursore nuovamente sull'istruzione DIM (la riga evidenziata), immettendo quindi record&, anziché record%. Quando il cursore non è più posizionato sulla riga, QBasic visualizza una finestra di dialogo che chiede: "Dopo questa modifica occorrerà riavviare il programma. Continuare?". Questo messaggio indica che la modifica apportata influisce sulla parte restante del programma, rendendo quindi necessario avviarlo nuovamente, se si desidera mantenere la modifica. Per accettarla, è sufficiente premere INVIO e quindi MAIUSC+F5 per avviare il programma dall'inizio. Dopo che i parametri sono stati immessi nuovamente, il programma visualizza il seguente messaggio:

```
File ordinato con successo
```

Se si esce da QBasic (selezionando il comando Esci dal menu File), è possibile utilizzare il comando TYPE del DOS per esaminare il file ESEMPIO.TST, scoprendo che non è stato eseguito alcun ordinamento e il file è rimasto invariato. Ora, è necessario caricare nuovamente QBasic e il programma ORDFILE.BAS.

Il problema non è ovvio, dato che QBasic non ha visualizzato alcun messaggio di errore. Ciò significa che si è verificato un errore logico, un difetto di progettazione nel sottoprogramma di ordinamento. Il modo migliore per affrontare questo problema consiste nell'eseguire il programma passo a passo, per controllare se è possibile individuare il punto errato. Dato che questa operazione richiede molto tempo, è opportuno eseguire il programma alla velocità normale, interrompendolo quando si arriva alla routine di ordinamento.

QBasic fornisce un aiuto, consentendo di impostare un *punto di interruzione*. Probabilmente la posizione migliore in cui interrompere il programma è la parte superiore del ciclo FOR...NEXT nel sottoprogramma OrdinaFile. Per posizionarsi in questo sottoprogramma è sufficiente premere F2, selezionare Ordinafile dalla casella di riepilogo e quindi premere INVIO. Poi, è necessario posizionare il cursore all'inizio del primo ciclo FOR...NEXT, premendo F9 per impostare un punto di interruzione. QBasic evidenzia la riga per indicare che il programma viene sospeso in quel punto.

Ora è necessario eseguire il programma, premendo MAIUSC+F5 e immettendo nuovamente i parametri. A questo punto viene visualizzato il messaggio: "File ordinato con successo".

La ragione di questo messaggio è che l'esecuzione non ha mai raggiunto il punto di interruzione. Evidentemente il programma non sta proprio eseguendo la routine di ordinamento. Si provi a inserire il punto di interruzione in una posizione diversa, ad esempio immediatamente sopra il ciclo principale di ordinamento; a questo scopo, occorre premere F9 per eliminare il punto di interruzione attuale, posizionando quindi il cursore alcune righe sopra, in corrispondenza dell'istruzione:

```
DO WHILE partizione% > 0
```

Premendo nuovamente F9, viene impostato un punto di interruzione su quella riga, che rappresenta l'inizio di tutto il ciclo di ordinamento e quindi deve essere raggiunta.

Per avviare nuovamente il programma, è sufficiente premere MAIUSC+F5. Ora il punto di interruzione ha funzionato, dato che il programma si è interrotto all'istruzione DO. Premendo F9, anche questo punto di interruzione viene cancellato. Ora, è possibile eseguire il ciclo passo a passo, una riga alla volta, premendo F8.

In effetti l'esecuzione, anziché andare all'interno del ciclo, è passata direttamente all'istruzione presente dopo il LOOP finale, che rappresenta un ciclo con una condizione d'entrata, quindi il test iniziale non viene eseguito correttamente.

Il ciclo viene eseguito soltanto quando la variabile partizione% contiene un valore maggiore di 0. Ora, è necessario posizionarsi nuovamente nell'istruzione DO e verificare il valore effettivo della variabile, utilizzando la tecnica della finestra Immediato, esaminata in precedenza.

Dopo aver verificato che il valore di partizione% è 0, si esamini la riga precedente. Il valore iniziale della variabile, ovvero la dimensione della partizione per l'ordinamento, viene ottenuto dividendo il numero dei record per 2; tuttavia, è nuovamente stato utilizzato il carattere di dichiarazione del tipo errato per la variabile record%, che deve invece essere un intero lungo.

Prima di cercare di eseguire di nuovo l'ordinamento del file, occorre verificare se questo stesso errore non è stato commesso anche in qualche altro punto. A questo scopo, è necessario effettuare un'operazione di ricerca e sostituzione, che può essere eseguita selezionando il comando Cambia del menu Cerca, premendo due volte ALT+C per selezionare l'opzione desiderata. QBasic visualizza una finestra di dialogo, in cui è possibile indicare la modifica da eseguire. Ora, è necessario digitare record% nel campo Trova, premere TAB per spostarsi nel campo Cambia in, immettendo record& come variabile da sostituire. Per avviare l'operazione di ricerca e sostituzione, è sufficiente premere INVIO. Quando QBasic visualizza una finestra di dialogo che indica che è stata individuato record%, è sufficiente premere INVIO per eseguire la modifica. In questo caso, è stato individuato soltanto un errore.

Ora, è necessario cercare di ordinare nuovamente il file, premendo MAIUSC+F5 per avviare il programma, che visualizza al termine il messaggio: "File ordinato con successo". Questa volta il tempo di esecuzione è stato superiore, ma la certezza che sia stato eseguito correttamente non c'è. Per verificarlo, è necessario salvare il file, e dopo essere usciti da QBasic utilizzare il comando TYPE per esaminarlo nuovamente. A prima vista sembra che l'ordinamento sia stato eseguito, ma esaminandolo più accuratamente è possibile notare che alcuni voci sono ancora fuori posto. Il programma ha ordinato parzialmente il file, ma non ha terminato il lavoro. A questo punto occorre ripristinare il file originale, copiando quello di backup ESEMPIO.DAT in ESEMPIO.TST nel seguente modo:

```
COPY ESEMPIO.DAT ESEMPIO.TST
```

Ora, occorre avviare nuovamente QBasic e caricare ORDFILE.BAS, eseguendo ancora l'ordinamento passo a passo. Prima però è necessario impostare di nuovo un punto di interruzione all'inizio dell'istruzione DO, in modo che quando il programma

viene avviato, si interrompa in quel punto. A questo scopo, è sufficiente premere F9 e quindi MAIUSC+F5 per avviare il programma e immettere i dati. Quando l'esecuzione si interrompe in corrispondenza del punto di interruzione, è sufficiente premere F9 per cancellarlo.

A questo punto, premendo F8 è possibile eseguire passo a passo la routine di ordinamento. Si noti che QBasic evidenzia l'istruzione attiva, mentre ci si sposta all'interno del programma. L'istruzione attuale è:

```
gamma% = record& - partizione%
```

che imposta la zona di ordinamento per il passaggio attuale nel file di dati in memoria.

Quando viene premuto nuovamente F8, il cursore si sposta all'inizio di un altro ciclo, che viene ripetuto fino a quando non sono stati scambiati tutti i dati fuori sequenza nella partizione attuale. L'istruzione successiva è:

```
scambio% = FALSO
```

Questa istruzione prepara la variabile scambio% per il passaggio successivo nella partizione attuale.

Si prema nuovamente F8 per passare all'istruzione:

```
FOR i% = 1 TO gamma%     'se necessario scambia gli elementi dell'array
```

Questa istruzione rappresenta l'inizio del ciclo più interno, che esegue tutto l'ordinamento.

Premendo nuovamente F8, l'esecuzione si interrompe all'istruzione successiva, che è il test che consente di determinare se una coppia di record deve essere rimpiazzata:

```
IF array(i%).sChiave > array(i% + partizione%).sChiave THEN
```

Potrebbe risultare utile verificare il contenuto di ogni record, quando viene sottoposto al test. Purtroppo, durante l'operazione di debug, QBasic fornisce soltanto la finestra Immediato per la visualizzazione dei valori delle variabili, anche se ne mantiene il contenuto, quando si passa alla finestra Edit. Dopo aver premuto F6 per tornare nuovamente alla finestra Immediato, è possibile eseguire le istruzioni già immesse, spostando semplicemente il cursore in corrispondenza della riga da eseguire e premendo INVIO.

Ora, all'interno della finestra Immediato, è possibile utilizzare l'istruzione PRINT per visualizzare il contenuto delle variabili array(i%).sChiave e array(i% + partizione%).sChiave. Si noti che il valore ASCII della variabile (ICIPA) è maggiore rispetto a quello della seconda variabile (BAB). Quindi, in base alla logica del programma, i valori delle due variabili dovrebbero essere stati scambiati utilizzando SWAP. Per verificare che l'operazione sia stata effettuata, è sufficiente premere F8 diverse volte per eseguire e spostarsi all'interno dell'istruzione IF.

Si passi nuovamente alla finestra Immediato e si eseguano ancora le istruzioni PRINT sulle variabili array(i%).sChiave e array(i% + partizione%).sChiave. La logica sembra funzionare correttamente e niente sembra essere errato. Ora, è necessario passare alla fine del ciclo FOR...NEXT per verificare se l'esecuzione prosegue anche al passaggio successivo. Dato che non è necessario esaminare passo a passo tutto il ciclo,

per accelerare l'analisi è possibile spostare il cursore sull'istruzione dopo NEXT i% alla fine del ciclo e quindi premere F9 per impostare un punto di interruzione e F5 per continuare.

A questo punto, si esegua nuovamente il programma passo a passo. L'istruzione attuale è:

```
gamma% = scambio%
```

La quale indica al programma di ordinare la partizione dei record che sono stati scambiati l'ultima volta. Il programma continua a spostarli, fino a quando non sono tutti in ordine.

Premendo F8, è possibile spostarsi all'istruzione successiva:

```
LOOP UNTIL NOT scambio%
```

che dovrebbe ricondurre nella parte superiore del ciclo interno DO...LOOP. Premendo nuovamente F8, il cursore si sposta in corrispondenza dell'istruzione:

```
partizione% = partizione% /2
```

in cui c'è qualcosa di errato, infatti l'esecuzione ha ignorato il ciclo, operazione che non avrebbe dovuto verificarsi fino a quando scambio% non fosse stata uguale a 0.

Utilizzando la finestra Immediato è possibile visualizzare il valore della variabile scambio%, che è 16, cioè un valore diverso da 0. Questo indica che il programma avrebbe dovuto continuare il ciclo.

Il test è corretto? Si verifichi che cosa esprime numericamente NOT scambio% mediante la finestra Immediato. Il problema è che QBasic ha valutato l'espressione come -17, che significa che questa variabile è vera (diverso da zero). Questa è la ragione per cui l'esecuzione non ha avuto un esito corretto.

Attualmente l'istruzione DO...LOOP viene ripetuta, fino a quando NOT scambio% è vero:

```
LOOP UNTIL NOT scambio%
```

Questa operazione è errata; per correggere questo errore, è sufficiente invertire la logica, in questo modo:

```
LOOP WHILE scambio%
```

Dopo aver immesso l'istruzione, è necessario premere F5 per avviare nuovamente il programma. Ancora una volta, viene visualizzato il messaggio: "File ordinato con successo". Per verificare che l'ordinamento sia stato eseguito correttamente, è sufficiente tornare al DOS e utilizzare il comando TYPE. A questo punto è possibile constatare che il file è stato correttamente ordinato, utilizzando il campo del codice del prodotto. Alla fine sono stati eliminati tutti gli errori del programma, grazie soprattutto all'interprete dell'ambiente QBasic; senza questo strumento, l'operazione di messa a punto del programma avrebbe richiesto un tempo decisamente superiore.

In questa parte sono stati esaminati molti degli strumenti forniti dall'ambiente di programmazione di QBasic per quanto riguarda la gestione e la messa a punto del programmi. Nella parte restante di questo capitolo, queste caratteristiche, e altre non ancora illustrate, vengono affrontate singolarmente e in modo più dettagliato.

Elimina ogni punto di interruzione

Si consulti anche

Punto di interruzione

Scopo

Il comando Elimina ogni punto di interruzione cancella tutti i punti di interruzione impostati, consentendo al programma di essere eseguito senza interruzioni; può essere utilizzato dopo che sono stati individuati e corretti tutti gli errori.

Selezione

Tastiera. Per aprire il menu Debug, occorre premere ALT+D. Il comando Elimina ogni punto di interruzione è il quinto; per selezionarlo, è sufficiente utilizzare i tasti di direzione per evidenziarlo, premendo quindi INVIO, oppure è possibile utilizzare la sequenza di tasti ALT+D+E.

Mouse. Si utilizzi il mouse per posizionare il cursore sul menu Debug, premendo quindi il pulsante sinistro. Quando il menu viene aperto, è sufficiente posizionare il cursore sul comando Elimina ogni punto di interruzione, premendo quindi nuovamente il pulsante sinistro del mouse.

Descrizione

I punti di interruzione sono i contrassegni che l'ambiente di QBasic inserisce nel codice per interrompere l'esecuzione del programma in corrispondenza di una determinata istruzione. Il comando Elimina ogni punto di interruzione consente di cancellare immediatamente tutti i punti di interruzione impostati durante una sessione di debug.

Procedura passo

Si consulti anche

Passo

Scopo

Il comando Procedura passo consente di eseguire l'istruzione successiva del programma QBasic, analizzando le chiamate alle procedure; può essere utilizzato quando si desidera esaminare in modo dettagliato il flusso di un programma.

Selezione

Tastiera. Per aprire il menu Debug, occorre premere ALT+D. Il comando Procedura passo è il secondo; per selezionarlo, è sufficiente utilizzare i tasti di direzione per evidenziarlo, premendo quindi INVIO, oppure è possibile utilizzare la sequenza di tasti ALT+D+R o il tasto funzione F10.

Mouse. Si utilizzi il mouse per posizionare il cursore sul menu Debug, premendo quindi il pulsante sinistro. Quando il menu viene aperto, è sufficiente posizionare il cursore sul comando Procedura passo, premendo quindi nuovamente il pulsante sinistro del mouse.

Descrizione

Il comando Procedura passo può essere utilizzato per eseguire il programma istruzione per istruzione. Generalmente, viene impostato un punto di interruzione all'interno di un programma in corrispondenza di un eventuale errore e quindi viene utilizzato questo comando per eseguire alcune istruzioni, verificando se l'errore si trova proprio in quel punto.

Quando questo comando è attivo, QBasic esegue le istruzioni passando brevemente alla finestra Output, se l'istruzione deve visualizzare alcuni dati, e ritornando quindi alla finestra Edit. Durante questa esecuzione passo a passo, è possibile apportare modifiche al programma oppure utilizzare altri comandi dei menu di QBasic.

Procedura passo consente di non controllare i sottoprogrammi o le funzioni il cui funzionamento è già stato accertato. Ad esempio, con l'istruzione:

```
CALL StampaSchermo
```

QBasic esegue il sottoprogramma StampaSchermo e continua con l'istruzione successiva a CALL. Questo comando viene generalmente utilizzato se è già stata eseguita la messa a punto dei sottoprogrammi e delle funzioni, se invece si desidera seguire l'esecuzione all'interno di un sottoprogramma o di una funzione, è necessario utilizzare il comando Passo, che consente di eseguire un'istruzione alla volta.

Imposta istruzione successiva

Scopo

Il comando Imposta istruzione successiva consente di riprendere l'esecuzione del programma (dopo un'interruzione temporanea) in corrispondenza di un'istruzione diversa da quella successiva in sequenza; può essere utilizzato per modificare il flusso normale di esecuzione, oppure per esaminare gli effetti provocati da passaggi effettuati alternativamente nell'istruzione IF o SELECT CASE.

Selezione

Tastiera. Per aprire il menu Debug, occorre premere ALT+D. Il comando Imposta istruzione successiva è l'ultimo; per selezionarlo, è sufficiente utilizzare i tasti di direzione per evidenziarlo, premendo quindi INVIO, oppure è possibile utilizzare la sequenza di tasti ALT+D+I o il tasto funzione F10.

Mouse. Si utilizzi il mouse per posizionare il cursore sul menu Debug, premendo quindi il pulsante sinistro. Quando il menu viene aperto, è sufficiente posizionare il cursore sul comando Imposta istruzione successiva, premendo quindi nuovamente il pulsante sinistro del mouse.

Descrizione

Il comando Imposta istruzione successiva consente di selezionare la prima istruzione da eseguire, quando il programma viene nuovamente avviato dopo essere stato sospeso da un punto di interruzione o dalla pressione di CTRL+INTERR. QBasic non esegue alcun codice presente tra l'ultima istruzione eseguita e quella nuova specificata, consentendo quindi di impostare qualsiasi variabile che la nuova sezione del codice richiede.

Per specificare una nuova istruzione, è sufficiente utilizzare i tasti di direzione (o il mouse) per posizionare il cursore sulla riga di programma corretta, selezionando quindi il comando Imposta istruzione successiva dal menu Debug. L'evidenziazione, che contrassegna l'istruzione attuale, si sposta in corrispondenza dell'istruzione selezionata e, quando il programma viene riavviato, l'esecuzione inizia in quel punto. Per eseguire quest'ultima operazione, è sufficiente premere F5 oppure F8, se si desidera eseguire le istruzioni successive passo a passo.

Passo

Si consulti anche

Procedura passo

Scopo

Il comando Passo consente di eseguire l'istruzione successiva di un programma QBasic; può essere utilizzato quando si desidera esaminare in modo dettagliato il flusso di un programma.

Selezione

Tastiera. Per aprire il menu Debug, occorre premere ALT+D. Dato che il comando Passo è il primo, per selezionarlo è sufficiente premere INVIO, oppure è possibile utilizzare la sequenza di tasti ALT+D+P o il tasto funzione F8.

Mouse. Si utilizzi il mouse per posizionare il cursore sul menu Debug, premendo quindi il pulsante sinistro. Quando il menu viene aperto, è sufficiente posizionare il cursore sul comando Passo, premendo quindi nuovamente il pulsante sinistro del mouse.

Descrizione

Il comando Passo può essere utilizzato per eseguire il programma istruzione per istruzione. Generalmente, viene impostato un punto di interruzione all'interno di un programma in corrispondenza di un eventuale errore e quindi viene utilizzato questo comando per eseguire alcune istruzioni, verificando se l'errore si trova proprio in quel punto.

Quando questo comando è attivo, QBasic esegue le istruzioni passando brevemente alla finestra Output se l'istruzione deve visualizzare alcuni dati e ritornando quindi alla finestra Edit. Durante questa esecuzione passo a passo, è possibile apportare modifiche al programma oppure utilizzare altri comandi dei menu di QBasic.

Il comando Passo consente di eseguire una alla volta le istruzioni di un sottoprogramma o di una funzione. Ad esempio, con l'istruzione:

```
CALL StampaSchermo
```

QBasic inserisce il sottoprogramma StampaSchermo nella finestra Edit e l'istruzione successiva da eseguire diventa la prima di quel sottoprogramma. Se non si desidera controllare i sottoprogrammi o le funzioni il cui funzionamento è già stato accertato, è possibile utilizzare il comando Procedura passo.

Punto di interruzione

Si consulti anche

Elimina ogni punto di interruzione

Scopo

Un punto di interruzione è una posizione specificata in cui l'esecuzione del programma si interrompe, durante la messa a punto. I punti di interruzione vengono impostati con lo scopo di sospendere l'esecuzione del programma prima di un punto particolarmente critico, oppure per interrompere l'esecuzione dopo che è già stata eseguita la procedura da verificare, esaminando il contenuto delle variabili.

Selezione

Tastiera. Per aprire il menu Debug, occorre premere ALT+D. Il comando Punto di interruzione è il quarto; per selezionarlo, è sufficiente utilizzare i tasti di direzione per evidenziarlo, premendo quindi INVIO, oppure è possibile utilizzare la sequenza di tasti ALT+D+P o il tasto funzione F9.

Mouse. Si utilizzi il mouse per posizionare il cursore sul menu Debug, premendo quindi il pulsante sinistro. Quando il menu viene aperto, è sufficiente posizionare il cursore sul comando Punto di interruzione, premendo quindi nuovamente il pulsante sinistro del mouse.

Descrizione

Il comando Punto di interruzione consente di attivare e disattivare un punto di interruzione. Selezionandolo la prima volta, viene inserito un punto di interruzione in corrispondenza della posizione del cursore; per eliminarlo, è sufficiente selezionare una seconda volta il comando, quando il cursore si trova nella stessa posizione. QBasic evidenzia qualsiasi riga che contenga un punto di interruzione.

Quando un programma individua un punto di interruzione sospende immediatamente l'esecuzione, consentendo quindi di esaminare le variabili attive o di sottoporre a test i valori di un'espressione, immettendo un'istruzione PRINT nella finestra Immediato per visualizzarli.

Al termine di questa verifica, per riprendere l'esecuzione del programma è sufficiente premere F5. È anche possibile utilizzare F8 per eseguire le istruzioni

successive passo a passo. Se il programma è ancora sospeso e si preme F9 una seconda volta per eliminare il punto di interruzione, l'esecuzione non riprende fino a quando non viene premuto F5. Se il programma contiene più di un punto di interruzione, è possibile cancellarli tutti contemporaneamente selezionando il comando Elimina ogni punto di interruzione dal menu Debug.

Analizza il flusso

Si consulti anche

TROFF, TRON

Scopo

Il comando Analizza il flusso attiva o disattiva l'analisi del programma nell'ambiente di QBasic. Quando questo comando è attivo, QBasic evidenzia ogni istruzione al momento dell'esecuzione, visualizzandola nella finestra Edit.

Selezione

Tastiera. Per aprire il menu Debug, occorre premere ALT+D. Il comando Analizza il flusso è il terzo; per selezionarlo, è sufficiente utilizzare i tasti di direzione per evidenziarlo, premendo quindi INVIO, oppure è possibile utilizzare la sequenza di tasti ALT+D+A.

Mouse. Si utilizzi il mouse per posizionare il cursore sul menu Debug, premendo quindi il pulsante sinistro. Quando il menu viene aperto, è sufficiente posizionare il cursore sul comando Analizza il flusso, premendo quindi nuovamente il pulsante sinistro del mouse.

Descrizione

Il comando Analizza il flusso attiva o disattiva l'analisi del programma; quando è attivo, appare un pallino accanto a esso nel menu Debug.

Questo comando consente a QBasic di esaminare il flusso di un programma durante l'esecuzione, permettendo di esaminarne i dettagli e di interromperlo (premendo CTRL+INTERR), nel caso non funzioni nel modo corretto. Qualsiasi punto di interruzione attivo prima della selezione di questo comando, continua a sospendere l'esecuzione in corrispondenza dei punti impostati. QBasic evidenza ogni istruzione al momento dell'esecuzione, consentendo di verificare se il programma viene eseguito nella sequenza corretta. Qualsiasi output generato dal programma viene visualizzato nella finestra Output, come al solito. Al termine di questa operazione, si ritorna alla finestra Edit.

Suggerimenti

Per fare in modo che sia il programma stesso ad attivare e disattivare l'analisi del flusso, è sufficiente inserire le istruzioni TRON e TROFF nei punti corretti del codice. Queste istruzioni possono essere utilizzate per limitare l'analisi a quella parte di programma in cui si ritiene siano presenti degli errori.

TRON e TROFF possono anche essere inserite direttamente nella finestra Immediato, operazione che ha gli stessi effetti provocati dall'attivazione e disattivazione del comando Analizza il flusso.

TROFF, TRON

Si consulti anche

Analizza il flusso

Scopo

L'istruzione TRON attiva il comando Analizza il flusso, che viene utilizzato per mettere a punto il programma. Quando questa caratteristica è attiva, QBasic evidenzia ogni istruzione al momento dell'esecuzione. L'istruzione TROFF permette di disattivare l'analisi.

Sintassi

```
TRON
TROFF
```

Utilizzo

```
TRON  'attiva l'analisi
FOR i% = 1 to gamma%
   IF array(i%) > array(i% + intervallo%) THEN
      SWAP array(i%), array(i% + intervallo%)
      scambio% = i%
   END IF
NEXT i%
TROFF 'disattiva l'analisi
```

Attiva l'analisi del flusso, esegue un ciclo FOR e quindi disattiva l'opzione.

Descrizione

Le istruzioni TROFF e TRON consentono di controllare l'esecuzione del programma nell'ambiente QBasic, come il comando Analizza il flusso.

TRON consente a QBasic di esaminare il flusso di un programma durante l'esecuzione. QBasic evidenza ogni istruzione al momento dell'esecuzione nella finestra Edit, permettendo di verificare se il programma viene eseguito nella sequenza corretta. Qualsiasi punto di interruzione attivo continua a sospendere l'esecuzione in corrispondenza dei punti impostati. Per interrompere il programma, è sufficiente premere CTRL+INTERR.

TROFF disattiva l'analisi del flusso, ma il programma continua l'esecuzione. Se queste due istruzioni vengono utilizzate insieme, consentono di isolare la parte di programma in cui si ritiene siano presenti degli errori.

TRON e TROFF possono anche essere immesse nella finestra Immediato per attivare direttamente l'analisi.

Esempio

Il programma di seguito riportato illustra l'utilizzo delle istruzioni TRON e TROFF, attivando e disattivando l'analisi, quando un utente preme un tasto. Generalmente è più semplice eseguire questa operazione selezionando il comando Analizza il flusso dal menu Debug.

```
CLS
cont& = 0
DO
  cont& = cont& + 1
  LOCATE 10, 10: PRINT cont&
  r$ = INKEY$: IF r$ = CHR$(27) THEN EXIT DO
  IF r$ <> "" THEN
     IF traccia% THEN
        traccia% = 0: TROFF 'disabilita l'analisi del flusso delle istruzioni
        LOCATE 11, 1: PRINT SPACE$(80);
        LOCATE 10, 10
     ELSE
        traccia% = -1: TRON  'abilita l'analisi del flusso delle istruzioni
     END IF
  END IF
LOOP WHILE 1
END
```

Appendice A

Messaggi di errore

I MESSAGGI DI ERRORE DI QBASIC

Gli errori possono verificarsi durante una delle tre seguenti fasi di sviluppo di un programma.

- Quando si avvia QBasic dalla riga di comando.
- Durante la modifica o la compilazione di un programma.
- Durante l'esecuzione di un programma.

Quando si verifica un errore, QBasic visualizza una finestra di dialogo che illustra il tipo di errore e consente di accedere alla finestra di Guida, che fornisce delle informazioni. In questa appendice viene riportato un elenco alfabetico e uno numerico, oltre alla spiegazione di ogni errore.

Errori di richiamo

Questi errori si verificano quando QBasic viene eseguito dalla riga di comando del DOS. La maggior parte degli errori è provocata dall'utilizzo errato delle opzioni.

Errori di sintassi e di compilazione

QBasic dispone di un editor che contiene tutte le regole di sintassi e che esamina il codice del programma durante l'immissione. Quando si cerca di spostare il cursore oltre la riga di programma corrente, l'editor verifica la sintassi e indica la presenza di eventuali errori, ad esempio la mancanza di parametri, variabili errate o parentesi non chiuse correttamente. Una finestra di dialogo visualizza un breve messaggio e consente di accedere alla Guida, che illustra in modo più dettagliato il tipo di errore.

Errori di esecuzione

Alcuni errori non si verificano fino al momento dell'esecuzione del programma, ad esempio quando un indice di un array va fuori limite, quando il programma cerca di dividere per 0, oppure nel caso in cui lo spazio dello stack disponibile non sia sufficiente per l'esecuzione. Se QBasic individua l'errore, visualizza una finestra di dialogo. Dato che la maggior parte degli errori di esecuzione corrisponde a un numero, le istruzioni ON ERROR GOTO sono in grado di rilevarli, permettendo in questo modo una gestione più semplice e rapida. L'istruzione ERR restituisce il numero o il codice dell'errore precedente. È anche possibile specificare un codice di errore nell'istruzione ERROR, in modo da simulare l'errore corrispondente. Nella Tabella A.1 sono riportati i codici di errore e la relativa descrizione.

In questa appendice sono elencati in ordine alfabetico tutti gli errori di richiamo, di compilazione e di esecuzione. Dato che questi tipi di messaggi di errore hanno un formato comune, questo elenco può essere facilmente utilizzato come riferimento. Ogni messaggio di errore è stampato in grassetto con accanto la specifica del tipo di errore (richiamo, compilazione o esecuzione) ed una spiegazione con alcuni suggerimenti per correggerlo. È necessario tenere presente che QBasic la maggior parte degli errori di compilazione, prima che il codice venga compilato. Infine, accanto ad ogni messaggio è indicato tra parentesi il numero o il codice dell'errore.

ERRORI DI RICHIAMO, DI COMPILAZIONE E DI ESECUZIONE

Blocco IF senza ENDIF
Errore di compilazione. In un'istruzione a blocchi IF...END IF, non corrisponde un END IF.

BYVAL ammette solo argomenti numerici
Errore di compilazione. Nella clausola BYVAL è possibile indicare soltanto variabili numeriche in una chiamata a una routine esterna.

Carattere di tipo non lecito in una costante numerica
Errore di compilazione. È stato utilizzato un carattere di dichiarazione del tipo errato, ad esempio 1.34%, in una costante numerica.

Carattere non valido
Errore di compilazione. Il file sorgente contiene un carattere non valido. Potrebbe essere stato inavvertitamente inserito un carattere di controllo, oppure si sta utilizzando un file proveniente da un programma di elaborazione testi che utilizza i codici di formattazione. Il file sorgente può contenere soltanto caratteri ASCII.

Caratteristica avanzata non disponibile
Errore di compilazione o di esecuzione. Si sta cercando di utilizzare una caratteristica di QBasic disponibile in un'altra versione del Basic o supportata soltanto in una versione del DOS successiva (73).

Tabella A.1 I codici di errore di QBasic

Codice	**Descrizione**
2	Errore di sintassi
3	RETURN senza GOSUB
4	Valori dell'istruzione DATA esauriti
5	Chiamata di funzione non lecita
6	Overflow
7	Memoria esaurita
9	Indice inferiore fuori limite
10	Definizione doppia
11	Divisione per zero
13	Tipo di dati non corrispondente
14	Spazio stringa esaurito
16	Formula a stringa troppo complessa
19	Manca RESUME
20	RESUME senza errore
24	Timeout di periferica
25	Errore di periferica
27	Carta esaurita
39	Era previsto CASE ELSE
40	È necessaria una variabile
50	Overflow dell'istruzione FIELD
51	Errore interno
52	Nome o numero del file errato
53	File non trovato
54	Modalità di accesso al file
55	File già aperto
56	Istruzione FIELD attiva
57	Errore di I/O su periferica
58	Il file esiste già
59	Lunghezza del record errata
61	Disco pieno
62	Input oltre la fine del file
63	Numero del record errato
64	Nome del file errato
67	Troppi file
68	Periferica non disponibile
69	Overflow nel buffer di comunicazioni
70	Permesso negato
71	Disco non pronto
72	Errore di supporto del disco
73	Caratteristica avanzata non disponibile
74	Tentativo di rinominare su altro disco
75	Errore di accesso al percorso/file
76	Percorso non trovato

Carta esaurita
Errore di esecuzione. È finita la carta della stampante oppure la stampante è spenta (27).

CASE senza SELECT
Errore di compilazione. Manca o è stata scritta in modo errato la prima parte di un'istruzione SELECT CASE.

Chiamata di funzione non lecita
Errore di esecuzione. A una funzione matematica o a una stringa è stato passato un parametro fuori intervallo (ad esempio, è stato specificato un numero negativo con la funzione SQR). Altri esempi comprendono indici di array troppo grandi o negativi (a meno che l'array non sia stato indicato correttamente con una clausola TO), numeri di record negativi oppure istruzioni di I/O eseguite su dispositivi errati o inesistenti (5).

Clausola lecita solo entro un blocco TYPE
Errore di compilazione. La clausola elemento AS tipo è ammessa solo all'interno di un blocco TYPE...END TYPE.

COMMON e DECLARE devono precedere le istruzioni eseguibili
Errore di compilazione. Le istruzioni COMMON o DECLARE devono precedere qualsiasi istruzione eseguibile. Le istruzioni che definiscono le condizioni delle variabili o del compilatore, oppure quelle che introducono commenti (COMMON, DEFtipo, DIM (per array statici), OPTION BASE, REM, TYPE...END TYPE) non sono eseguibili. Anche i metacomandi non sono eseguibili.

CONST/DIM SHARED segue SUB/FUNCTION
Errore di compilazione. Le istruzioni CONST e DIM SHARED vanno anteposte a qualsiasi definizione di sottoprogramma o di procedura FUNCTION. Questo messaggio potrebbe provocare alcuni problemi, anche se il programma viene eseguito correttamente.

Costante non valida
Errore di compilazione. È stata utilizzata un'espressione non valida nell'assegnare un valore a una costante. Le espressioni numeriche possono contenere soltanto costanti numeriche o letterali e qualsiasi valore numerico, tranne l'elevamento a potenza (^). Un'espressione stringa assegnata a una costante deve sempre essere un'unica stringa letterale.

DEF FN non ammessa nelle istruzioni di controllo
Errore di compilazione. Non è possibile definire una funzione con l'istruzione DEF FN all'interno di una struttura di controllo (ad esempio IF o SELECT CASE).

DEF senza END DEF
Errore di compilazione. Manca l'istruzione END DEF nella definizione di una funzione multiriga.

Definizione di funzione nidificata
Errore di compilazione. Si è cercato di inserire un'istruzione FUNCTION all'interno di un'altra istruzione FUNCTION, oppure all'interno dell'istruzione IF.

Deve essere la prima istruzione della riga
Errore di compilazione. Le istruzioni a blocco IF...THEN...ELSE, IF, ELSE, ELSEIF e END IF possono essere precedute solo da un numero o da un'etichetta di riga.

Disco non pronto
Errore di esecuzione. Lo sportello dell'unità disco è aperto, oppure non è stato inserito il disco nell'unità (71).

Disco pieno
Errore di esecuzione. Lo spazio disponibile su disco non è sufficiente per portare a termine un'operazione PRINT, WRITE o CLOSE (61).

Divisione per zero
Errore di compilazione o di esecuzione. Il programma ha incontrato una divisione per zero. Probabilmente, non è stata inizializzata una variabile o il nome è stato scritto in modo errato (11).

DO senza LOOP
Errore di compilazione. Ogni istruzione DO deve terminare con la parola chiave LOOP.

Documento troppo grande
Editor. Il documento supera il limite interno dell'editor di QBasic; è necessario suddividerlo in file separati.

Definizione doppia
Errore di compilazione o di esecuzione. Si sta utilizzando un identificatore già definito. Non è possibile utilizzare costanti, variabili, sottoprogrammi o funzioni che abbiano lo stesso nome. Questo errore può verificarsi anche quando si cerca di ridimensionare un array dinamico utilizzando una seconda istruzione DIM. Per eseguire questa operazione, è necessario utilizzare ERASE e DIM (o soltanto REDIM) (10).

Duplice etichetta
Errore di compilazione. È stata utilizzata la stessa etichetta (o numero di riga) per più di una riga di programma.

È necessaria l'assegnazione di un record/stringa
Errore di compilazione. È stata utilizzata un'istruzione LSET, senza specificare un'assegnazione a una stringa o a una variabile record.

È necessaria l'assegnazione di una stringa
Errore di compilazione. È stata utilizzata un'istruzione RSET, senza specificare un'assegnazione a una stringa.

È necessaria un'espressione a stringa
Errore di compilazione. La sintassi dell'istruzione richiede un'espressione stringa. Probabilmente, è stata utilizzata una variabile numerica o un letterale numerico. Generalmente, quando QBasic richiede un'espressione stringa, è possibile sfruttare qualsiasi combinazione di variabili stringa, costanti stringa e letterali stringa.

È necessaria una clausola AS
Errore di compilazione. È stata dichiarata una variabile utilizzando una clausola AS e quindi quella variabile è stata specificata in un'istruzione DIM, REDIM SHARED o COMMON senza includere la clausola AS.

È necessaria una clausola AS alla prima dichiarazione
Errore di compilazione. Si è fatto riferimento al nome di una variabile con la clausola AS, che però non è stata inserita nella dichiarazione originale. Si aggiunga la clausola alla dichiarazione.

È necessaria una stringa a lunghezza variabile
Errore di compilazione. Si è cercato di utilizzare una stringa a lunghezza fissa in un'istruzione FIELD, operazione non consentita in QBasic.

È necessaria una variabile
Errore di compilazione o di esecuzione. Durante la compilazione, è stata utilizzata un'istruzione INPUT, LET, READ o SHARED, senza specificare il nome della variabile. Al momento dell'esecuzione, è stata utilizzata l'istruzione GET o PUT senza specificare una variabile, quando il file è stato aperto in modalità binaria (40).

È necessaria una variabile a stringa
Errore di compilazione. L'istruzione richiede come argomento una variabile stringa.

È necessario DECLARE
Errore di compilazione. È stata eseguita una subroutine o una chiamata di funzione implicita (assegnando il nome direttamente nell'istruzione, anziché utilizzando l'istruzione CALL), che non sono state dichiarate prima di richiamarle.

È necessario un intero tra 1 e 32767
Errore di compilazione. L'istruzione richiede un argomento di tipo intero positivo.

È prevista una variabile semplice o di matrice
Errore di compilazione. QBasic richiede come argomento una variabile; è possibile utilizzare una variabile semplice o una matrice.

È previsto GOTO o GOSUB
Errore di compilazione. La sintassi richiede un'istruzione GOTO o una GOSUB.

È previsto un identificatore
Errore di compilazione. È stato utilizzato un identificatore errato (ad esempio un numero o una parola chiave riservata).

Elemento di matrice dinamica non valido
Errore di compilazione. Si è cercato di utilizzare VARPTR$ con un elemento di array dinamico. Dato che l'indirizzo di questo elemento potrebbe cambiare durante l'esecuzione del programma, non è possibile farvi affidamento.

Elemento non definito
Errore di compilazione. Si è fatto riferimento a un campo di un tipo definito dall'utente, senza aver definito il campo nell'istruzione TYPE...END TYPE.

ELSE senza IF
Errore di compilazione. A una clausola ELSE manca la corrispondente clausola IF, oppure è stata cancellata.

ELSEIF senza IF
Errore di compilazione. A una clausola ELSEIF manca la corrispondente clausola IF, oppure è stata cancellata.

END DEF senza DEF
Errore di compilazione. A un'istruzione END DEF non corrisponde alcuna istruzione DEF, oppure è stata cancellata.

END IF senza un blocco IF
Errore di compilazione. Manca l'inizio di un blocco IF.

END SELECT senza SELECT
Errore di compilazione. La prima parte di un'istruzione SELECT CASE è mancante o è stata immessa in modo errato.

END SUB/FUNCTION deve essere l'ultima riga della finestra
Errore di compilazione. Si è tentato di aggiungere un codice dopo una procedura. È necessario ritornare al modulo principale o aprirne un altro.

END SUB/FUNCTION senza SUB/FUNCTION
Errore di compilazione. A un'istruzione END SUB/FUNCTION non corrisponde alcuna parola chiave SUB o FUNCTION, oppure sono state cancellate.

END TYPE senza TYPE
Errore di compilazione. Un'istruzione END TYPE è stata utilizzata al di fuori di una dichiarazione TYPE, oppure è stata cancellata la clausola TYPE.

Era previsto (a): voce
Errore di compilazione. La sintassi dell'istruzione richiede la voce specificata (ad esempio una parentesi di chiusura o una virgola).

Errore di accesso al percorso/file
Errore di compilazione o di esecuzione. Nel corso di un'operazione OPEN, MKDIR, CHDIR o RMDIR il DOS non è stato in grado di collegare il nome del percorso di ricerca

con quello del file. Si controlli il nome del percorso e (se si sta cercando di scrivere o cancellare un file) gli attributi del file (75).

Errore di periferica
Errore di esecuzione. Una periferica ha inviato un messaggio di errore hardware. Se questo errore si è verificato durante la trasmissione di dati a un file di comunicazione, significa che i segnali specificati nell'istruzione OPEN COM non sono stati ricevuti nel periodo specificato (25).

Errore di $Metacomando
Errore di compilazione. Il metacomando specificato è errato; l'ortografia potrebbe essere non corretta.

Errore di caricamento del file (nomefile) - Errore di I/O su disco
Errore di richiamo. Si è cercato di caricare un file in QBasic, ma sono sorti dei problemi con l'unità disco (ad esempio lo sportello dell'unità viene lasciato aperto).

Errore di caricamento del file (nomefile) - Errore nella zona memoria del DOS
Errore di richiamo. Si è cercato di caricare un file in QBasic, ma la parte di memoria necessaria al DOS era stata sovrascritta con una routine in linguaggio assembler, oppure per mezzo dell'istruzione POKE.

Errore di caricamento del file (nomefile) - Impossibile trovare il file
Errore di richiamo. Si è cercato di dirigere nuovamente l'input da un file a QBasic, ma il file non si trova nel punto specificato sulla riga di comando.

Errore di caricamento del file (nomefile) - Memoria esaurita
Errore di esecuzione. Un'operazione richiede più memoria di quella disponibile (ad esempio per allocare un buffer del file). Si cerchi di ridurre le dimensioni dei buffer DOS, di eliminare i programmi residenti in memoria oppure qualche driver di periferica, di utilizzare array dinamici, invece di quelli statici. La chiusura di eventuali documenti può liberare una parte di spazio.

Errore di sintassi
Errore di compilazione o di esecuzione. Al momento della compilazione, questo messaggio di solito indica un errore ortografico o una punteggiatura errata. QBasic indica questo errore non appena si cerca di spostare il cursore dalla riga corrente. Al momento dell'esecuzione, un errore di sintassi indica generalmente una voce formattata in modo errato nell'istruzione DATA (2).

Errore di sintassi in una costante numerica
Errore di compilazione. Una costante numerica non ha la forma corretta.

Errore di sottoprogramma
Questo errore si verifica quando si cerca di definire un sottoprogramma o una funzione già definiti. Si provi a inserire la definizione al'interno di un'altra, oppure si tralasci l'istruzione END SUB o END FUNCTION.

Errore di supporto del disco
Errore di esecuzione. L'hardware dell'unità ha rilevato un difetto fisico sul disco. Si potrebbe impedire l'uso del settore errato, utilizzando il comando FORMAT del DOS, dopo aver effettuato il backup di tutti i dati che possono essere ripristinati.

Errore durante l'inizializzazione di QBasic
Errore di richiamo. Generalmente, indica che la memoria disponibile non è sufficiente per caricare QBasic. Si cerchi di eliminare i programmi residenti in memoria non necessari. Questo errore potrebbe verificarsi anche se il proprio computer non è completamente IBM compatibile.

Errore I/O su periferica
Errore di esecuzione. Una periferica ha segnalato un errore di I/O e il sistema operativo non è in grado di proseguire dall'errore (57).

Errore non visualizzabile
Errore di esecuzione. Non è disponibile un messaggio di errore relativo alla condizione esistente. La causa può essere un'istruzione ERROR con un codice di errore non definito.

Espressione troppo complessa
Errore di compilazione. È stata creata un'espressione con troppi elementi o i cui elementi sono troppo nidificati. L'espressione potrebbe richiedere troppe stringhe temporanee per contenere i letterali. Si cerchi di semplificare l'espressione, ad esempio assegnando le stringhe alle variabili e utilizzando queste ultime, invece delle stringhe letterali.

Etichetta non definita
Errore di compilazione. È stata specificata un'etichetta di riga in un'istruzione GOTO o GOSUB, ma l'etichetta non esiste.

Etichetta non definita: etichetta
Errore di compilazione. Un'istruzione GOTO ha fatto riferimento a un'etichetta di riga inesistente.

EXIT DO esterno a un ciclo DO...LOOP
Errore di compilazione. Un'istruzione EXIT DO è stata utilizzata all'esterno di un'istruzione DO...LOOP.

EXIT esterno a un ciclo FOR...NEXT
Errore di compilazione. Un'istruzione EXIT FOR è stata utilizzata all'esterno di un'istruzione FOR...NEXT.

File già aperto
Errore di esecuzione. Si è cercato di aprire un file per l'output sequenziale, ma il file era già aperto, oppure si è cercato di utilizzare KILL per cancellare un file aperto (55).

File già caricato
Errore di compilazione. Si è cercato di caricare un file già in memoria.

File non trovato
Errore di esecuzione. Un'istruzione FILES, KILL, NAME, OPEN o RUN fa riferimento a un file che non esiste (53).

File sorgente ad accesso binario
Errore di compilazione. Si è cercato di compilare un file binario, anziché un file sorgente ASCII. Se si desidera compilare un file sorgente BASICA, è necessario innanzitutto salvarlo con l'opzione ,A.

Fine file imprevista in una dichiarazione TYPE
Errore di compilazione. In un blocco TYPE...END TYPE è presente un carattere di fine file. Se si dispone di un editor in grado di ricercare i caratteri di controllo, è possibile cercare e cancellare i caratteri CTRL+Z.

FOR senza NEXT
Errore di compilazione. A ciascuna istruzione FOR deve corrispondere un'istruzione NEXT.

Formula a stringa troppo complessa
Errore di esecuzione. È stata utilizzata una formula a stringa troppo lunga oppure sono state specificate più di 15 variabili in un'istruzione INPUT. Per eseguire correttamente il programma, è necessario scomporre la formula o l'istruzione INPUT in più parti (16).

Funzione già definita
Errore di compilazione. Si è cercato di definire una funzione già definita.

Funzione non definita
Errore di compilazione. Si è cercato di utilizzare una funzione definita dall'utente, prima di averla definita o dichiarata.

Guida non trovata
Editor. QBasic non è stato in grado di fornire la Guida, dato che errori di programma impediscono di costruire una tabella di variabili. Premendo F5, è possibile visualizzare la riga che ha provocato l'errore.

Identificatore troppo lungo
Errore di compilazione. Si è cercato di utilizzare un identificatore con una lunghezza superiore ai 40 caratteri.

Il file esiste già
Errore di esecuzione. Si è cercato di specificare un file in un'istruzione NAME, ma il nome specificato è analogo a quello di un file o di una directory esistenti sull'unità disco in uso (58).

Impossibile continuare
Errore di esecuzione. Durante la messa a punto, è stata eseguita una modifica che impedisce la continuazione dell'esecuzione.

Indice inferiore fuori limite
Errore di esecuzione. Il programma ha cercato di fare riferimento a un elemento di array con un indice che ne superava le dimensioni oppure si è cercato di accedere a un elemento di un array dinamico non dimensionato (ad esempio, se l'array è stato cancellato) oppure si è cercato di utilizzare un array che supera i 64 KB (9).

Input oltre la fine del file
Errore di esecuzione. Un'istruzione INPUT ha tentato di leggere da un file vuoto o da un file dal quale sono già stati letti tutti i dati. Per evitare questo errore, si utilizzi la funzione EOF per rilevare la fine del file (62).

Istruzione FIELD attiva
Errore di esecuzione. Un'istruzione GET o PUT ha specificato una variabile di record per un file in cui lo spazio era stato allocato dall'istruzione FIELD. GET e PUT non possono essere utilizzate con una variabile di record specificata in un'istruzione FIELD (56).

Istruzione lecita solo entro una SUB, FUNCTION o DEF FN
Errore di compilazione. Questa istruzione non è lecita nel codice al livello del modulo.

Istruzione lecita solo entro una SUB/FUNCTION
Errore di compilazione. Questa istruzione non è lecita nel codice al livello del modulo.

Istruzione non lecita in modalità diretta
Errore di compilazione. L'istruzione è valida solo all'interno di un programma e non può essere utilizzata nella finestra Immediato.

Istruzione non lecita in un blocco TYPE
Errore di compilazione. All'interno dell'istruzione TYPE...END TYPE, è possibile utilizzare soltanto REM o la clausola elemento AS nometipo.

Istruzione non lecita in una procedura o una DEF FN
Errore di compilazione. L'istruzione non può essere utilizzata all'interno di un'istruzione FUNCTION, SUB o DEF FN.

Istruzione TYPE nidificata in modo errato
Errore di compilazione. Non è possibile inserire la definizione di un tipo definito dall'utente all'interno di un sottoprogramma o di una funzione.

L'istruzione non può precedere le definizioni SUB/FUNCTION
Errore di compilazione. REM e DEFtipo sono le uniche due istruzioni ammesse prima della definizione di una subroutine di una funzione.

L'operazione prevede un disco
Errore di compilazione. È stata specificata un'operazione di lettura o di scrittura che richiede una periferica su disco, non una tastiera, una stampante.

LOOP senza DO
Errore di compilazione. È stata utilizzata la parola chiave LOOP, senza però iniziare la struttura con la parola chiave DO.

Lunghezza del record errata
Errore di esecuzione. È stata eseguita un'istruzione GET o PUT, che ha specificato una variabile di record la cui lunghezza non corrisponde a quella del record indicata nella relativa istruzione OPEN (59).

Manca AS
Errore di compilazione. L'istruzione utilizzata richiede la parola chiave AS.

Manca BASE
Errore di compilazione. È stata utilizzata la parola OPTION, senza farla seguire dalla parola chiave BASE.

Manca GOSUB
Errore di compilazione. Ad un'istruzione ON evento, manca GOSUB.

Manca GOTO
Errore di compilazione. Ad un'istruzione ERROR, manca GOTO.

Manca input
Errore di compilazione. La sintassi dell'istruzione richiede la parola chiave INPUT.

Manca NEXT per variabile
Errore di compilazione. La parola chiave NEXT non è stata utilizzata in un ciclo FOR. variabile è la variabile di conteggio del ciclo.

Manca RESUME
Errore di esecuzione. Il programma è terminato durante l'esecuzione di una routine di gestione degli errori. Per restituire il controllo al gestore degli errori, è sufficiente inserire l'istruzione RESUME (19).

Manca SUB o FUNCTION
Errore di compilazione. A un'istruzione DECLARE non corrisponde alcuna procedura.

Manca THEN
Errore di compilazione. Si è iniziato a scrivere un'istruzione IF, dimenticando però di inserire la parola chiave THEN.

Manca TO
Errore di compilazione. Generalmente, questo messaggio significa che si è cercato di specificare un intervallo degli indici di un array senza inserire la parola chiave TO.

Manca TYPE
Errore di compilazione. In un'istruzione END TYPE, manca la parola chiave TYPE.

Manca un asterisco
Errore di compilazione. Si è cercato di dichiarare una stringa a lunghezza fissa, senza però inserire l'asterisco prima del numero di caratteri.

Manca un punto e virgola
Errore di compilazione. La sintassi dell'istruzione richiede un punto e virgola.

Manca un segno di sottrazione
Errore di compilazione. La sintassi dell'istruzione richiede un segno di sottrazione.

Manca un segno di uguale
Errore di compilazione. La sintassi dell'istruzione richiede un segno di uguale (ad esempio in un'istruzione di assegnazione).

Manca un'etichetta o un numero di riga
Errore di compilazione. È stata utilizzata un'istruzione che richiede un numero di riga o un'etichetta.

Manca una parentesi destra
Errore di compilazione. La sintassi dell'istruzione richiede una parentesi destra o di chiusura.

Manca una parentesi sinistra
Errore di compilazione. La sintassi dell'istruzione richiede una parentesi sinistra, oppure l'istruzione REDIM ha cercato di riallocare lo spazio per una variabile che non è un array.

Manca una virgola
Errore di compilazione. La sintassi dell'istruzione richiede una virgola.

Mancata corrispondenza dei tipi
Errore di compilazione o di esecuzione. La variabile utilizzata non è del tipo richiesto.

Matrice già dimensionata
Errore di compilazione o di esecuzione. È stata utilizzata più di un'istruzione DIM con un array statico. Questo errore si verifica anche se viene utilizzata l'istruzione OPTION BASE dopo aver dimensionato un array. È necessario controllare il tipo di array (statico o dinamico) e la posizione dell'istruzione OPTION BASE (se presente).

Matrice non definita
Errore di compilazione. Si è fatto riferimento a un array con più di 10 elementi, senza averlo prima definito. Se l'array non è stato definito in nessun altro punto, è sufficiente aggiungere l'istruzione DIM corretta. Se invece l'array si trova in un altro modulo, è necessario utilizzare l'istruzione appropriata (ad esempio COMMON, SHARED e così via).

Matrice numerica non lecita
Errore di compilazione. Gli array numerici non sono ammessi come argomenti di VARPTR$, ma sono consentite solo le variabili semplici e gli array di tipo stringa.

Matrice troppo grande
Errore di compilazione. Si è cercato di dimensionare un array statico che supera i 64 KB, oppure si è esaurito lo spazio di memoria disponibile nel segmento dati di default (DGROUP). Si cerchi di ridurre le dimensioni dell'array o dei requisiti dei dati (ad esempio le stringhe), oppure si dichiari l'array dinamico, in modo che possa essere utilizzato nel far heap.

Memoria a distanza alterata
Errore di compilazione. Questo messaggio potrebbe essere provocato da un programma residente in memoria che entra in conflitto con QBasic, da un'istruzione POKE oppure da una routine non Basic, che ha inavvertitamente modificato le aree di memoria utilizzate da QBasic.

Memoria esaurita
Errore di richiamo, di compilazione o di esecuzione. Un'operazione richiede più memoria di quella disponibile (ad esempio per allocare il buffer di un file). Si cerchi di ridurre le dimensioni dei buffer DOS, di eliminare i programmi residenti in memoria oppure qualche driver di periferica. La chiusura di eventuali documenti può liberare una parte di spazio (7).

Modalità di accesso al file errata
Errore di esecuzione. Questo errore si verifica nelle seguenti circostanze:

- Quando le istruzioni PUT o GET vengono utilizzate con un file ad accesso sequenziale (54).
- Quando si cerca di eseguire l'istruzione OPEN con una modalità diversa da I, O o R (54).
- Quando il programma cerca di utilizzare un'istruzione FIELD in un file non aperto per l'accesso casuale.
- Quando il programma cerca di scrivere in un file aperto per l'input.
- Quando il programma cerca di leggere da un file aperto per l'output o per l'accodamento.
- Quando si cerca di caricare un programma binario modificato.

NEXT senza FOR
Errore di compilazione. A ciascuna istruzione NEXT deve corrispondere un'istruzione FOR.

Nome del file errato
Errore di compilazione o di esecuzione. Il nome del file specificato ha violato una delle regole per i file DOS (caratteri non validi, nome troppo lungo e così via). Questo errore potrebbe verificarsi anche se si inserisce un errore di sintassi in un'istruzione OPEN, ad esempio se vengono omessi i doppi apici nel nome del file (64).

Nome di funzione non valido
Errore di compilazione. Una parola riservata è stata utilizzata come nome di una FUNCTION definita dall'utente.

Nome di sottoprogramma non valido
Errore di compilazione. È stata utilizzata una parola riservata come nome di un sottoprogramma, oppure lo stesso nome è stato sfruttato per due sottoprogrammi diversi.

Nome o numero del file errato
Errore di esecuzione. Un'istruzione o un comando fanno riferimento a un file con un nome o un numero non specificato nell'istruzione OPEN, oppure è stato utilizzato un numero al di fuori dell'intervallo specificato (52).

Non corrisponde il numero degli argomenti
Errore di compilazione. Il numero di argomenti utilizzato in un sottoprogramma o in una funzione è errato. Si confronti la chiamata alla funzione o al sottoprogramma con la riga di intestazione o la dichiarazione.

Non corrispondono i tipi dei parametri
Errore di compilazione. Un parametro di un sottoprogramma o di una funzione non ha lo stesso tipo del corrispondente argomento nell'istruzione DECLARE o nella chiamata.

Non può iniziare per 'FN'
Errore di compilazione. Non è possibile iniziare il nome di una variabile, di un sottoprogramma o di una funzione, definita da FUNCTION, con le lettere FN, che possono essere utilizzate soltanto per le funzioni definite con l'istruzione DEF FN.

Numero del record errato
Errore di esecuzione. In un'istruzione PUT o GET, il numero di un record è minore o uguale a 0 (questo messaggio si presenta, ad esempio, quando la variabile non è stata inizializzata) (63).

Numero di dimensioni errato
Errore di compilazione. Si è fatto riferimento a un elemento di array, utilizzando un numero di dimensioni diverso da quello specificato nella definizione dell'array.

Numero non lecito
Errore di compilazione. Il formato del numero non è valido; probabilmente sono stati inseriti caratteri non numerici.

Overflow
Errore di esecuzione. Il risultato di un calcolo è troppo elevato per poter essere rappresentato nell'intervallo consentito per i numeri in virgola mobile o interi (6).

Overflow dell'istruzione FIELD
Errore di esecuzione. Un'istruzione FIELD ha tentato di allocare più byte di quanti erano stati specificati per la lunghezza dei record di un file ad accesso casuale (50).

Overflow della memoria dati
Errore di compilazione. La memoria utilizzata per gli elementi degli array statici, per le stringhe o per altre variabili è superiore a quella disponibile. Si cerchi di rendere gli array dinamici, se possibile, oppure di disattivare le opzioni di messa a punto. Se lo spazio in memoria rimane insufficiente, è necessario scomporre il programma in diverse parti, che possono essere poi concatenate.

Overflow della memoria programma
Errore di compilazione. Si è cercato di compilare un programma, il cui segmento di codice supera i 64 KB. È necessario suddividere il programma in più moduli o utilizzare l'istruzione CHAIN.

Overflow in una costante numerica
Errore di compilazione. La costante numerica utilizzata è troppo grande per il tipo di numeri rappresentato (ad esempio un intero superiore a 32767).

Overflow matematico
Errore di compilazione. Il risultato di un calcolo è troppo grande per poter essere rappresentato nel formato numerico di QBasic.

Overflow nel buffer comunicazioni
Errore di esecuzione. Durante le comunicazioni a distanza, è stata superata la capacità del buffer di comunicazione. Si cerchi di aumentarne le dimensioni, aumentando il valore specificato nel parametro RB nell'istruzione OPEN COM, verificando il buffer con più frequenza (mediante la funzione LOC) per individuare possibili overflow, oppure leggendo il buffer più spesso con la funzione INPUT$ (69).

Parametri formali ripetuti
Errore di compilazione. Lo stesso nome non può essere utilizzato più di una volta in un elenco dei parametri.

Percorso non trovato
Errore di esecuzione. Nel corso di un'operazione OPEN, MKDIR, CHDIR o RMDIR il DOS non è stato in grado di trovare il percorso di ricerca specificato (76).

Periferica non disponibile.
Errore di esecuzione. La periferica alla quale si è tentato di accedere non è in linea oppure non esiste (68).

Permesso negato
Errore di esecuzione. Si è tentato di scrivere su un disco a sola lettura o di accedere a un file bloccato (70).

Procedura troppo grande
Errore di compilazione. La procedura ha superato il limite interno di QBasic. Per utilizzarla, è necessario suddividerla in diverse procedure più piccole.

Programma troppo esteso a livello di modulo
Errore di compilazione. Il codice a livello di modulo supera il limite interno di QBasic. Si cerchi di spostare una parte del codice all'interno di una procedura SUB o FUNCTION.

RESUME senza errore
Errore di esecuzione. Un'istruzione RESUME è stata inserita all'esterno di una routine di gestione degli errori (20).

RETURN senza GOSUB
Errore di esecuzione. Il programma ha incontrato un'istruzione RETURN alla quale non corrisponde una precedente istruzione GOSUB (3).

Ricominciare da capo
Errore di esecuzione. La risposta a un'istruzione INPUT conteneva numero o tipo di voci di dati errato. La risposta deve essere nuovamente immessa, utilizzando il formato corretto.

Riga troppo lunga
Errore di compilazione. Le righe devono contenere un massimo di 255 caratteri.

Salto alla successiva istruzione END TYPE
Errore di compilazione. Un errore nell'istruzione TYPE ha fatto ignorare a QBasic tutto ciò che si trova tra le istruzioni TYPE ed END TYPE. Anche se la compilazione prosegue, il programma probabilmente non funziona correttamente.

SELECT senza END SELECT
Errore di compilazione. Manca o è stata immessa in modo errato la fine di un'istruzione SELECT CASE.

Separatore non valido
Errore di compilazione. È stato utilizzato un carattere diverso da un punto e virgola o da una virgola come separatore in un'istruzione PRINT USING o WRITE.

Sintassi dell'indice non valida
Errore di compilazione. Si è verificato un errore nella specifica dell'indice di un array (ad esempio, si è cercato di utilizzare un valore stringa).

Sono permesse solo variabili semplici
Errore di compilazione. I tipi utente e gli array non sono consentiti nelle istruzioni READ e INPUT. Sono ammesse solo variabili semplici ed elementi di array non di tipo utente.

Sottoprogramma non definito
Errore di compilazione. È stato chiamato un programma non definito. Potrebbe essere necessario caricare il modulo di programma o la libreria che contiene la definizione.

Spazio dati esaurito
Errore di compilazione o di esecuzione. Si è esaurito lo spazio nel segmento dati di default (DGROUP), che contiene le stringhe a lunghezza fissa e le variabili semplici. Si cerchi di ridurre il buffer o lo stack, di utilizzare tipi di dati (ad esempio interi) che non richiedono più spazio di quanto necessario oppure di prendere in considerazione l'utilizzo di algoritmi alternativi.

Spazio stack esaurito
Errore di esecuzione. Il programma contiene troppi sottoprogrammi o funzioni attive, oppure una funzione ricorsiva ha chiamato se stessa troppe volte. Questo errore non può essere intercettato.

Spazio stringa esaurito
Errore di esecuzione. Le variabili a stringa superano la capacità dello spazio destinato alle stringhe. Si cerchi di utilizzare array dinamici di stringhe a lunghezza fissa (14).

Spazio stringhe alterato
Errore di esecuzione. Questo errore può essere provocato da uno dei seguenti problemi:

- Un'istruzione POKE o DEF SEG errata ha modificato un descrittore di stringa o un puntatore.
- Si è verificata una mancata corrispondenza tra due programmi concatenati nelle dichiarazioni COMMON.

Specifica del carattere DEFxxx non valida
Errore di compilazione. Non è stata digitata correttamente un'istruzione DEFtipo. I tipi validi sono INT, SNG, DBL, LGN e STR oppure (per funzioni definite dall'utente) uno spazio.

Specifica del parametro formale non valida
Errore di compilazione. C'è un errore nell'elenco dei parametri di una funzione o di un sottoprogramma.

Stringa a lunghezza fissa non lecita
Errore di compilazione. Si è tentato di utilizzare una stringa a lunghezza fissa come parametro formale.

Struttura di controllo in IF...THEN...ELSE incompleta
Errore di compilazione. In un'istruzione su una sola riga IF...THEN...ELSE appare un'istruzione END IF, END SELECT, LOOP, NEXT o WEND senza corrispondente.

SUB/FUNCTION senza END SUB/FUNCTION
Errore di compilazione. A una procedura manca l'istruzione finale.

Tentativo di rinominare su altro disco
Errore di esecuzione. Si è tentato di assegnare un nuovo nome a un file, cambiandone l'unità disco, operazione non consentita in QBasic. Si copi il file nella nuova unità disco, utilizzando quindi KILL per cancellare il file originale (74).

Timeout di periferica
Errore di esecuzione. Durante un'operazione di I/O su periferica, il programma non ha ricevuto le informazioni nel periodo di tempo specificato (24).

Tipo non definito
Errore di compilazione. L'argomento tipo utente di un'istruzione TYPE non è stato definito. Potrebbe essere necessario caricare il modulo o la libreria che contiene la definizione.

Tipo utente supera i 65535 byte
Errore di compilazione. Un tipo definito dall'utente non può superare i 64 KB.

Troppe definizioni TYPE
Errore di compilazione. Il numero massimo consentito di tipi definiti dall'utente è 240.

Troppe dimensioni
Errore di compilazione. Si è cercato di specificare più di 60 dimensioni nella definizione di un array.

Troppe etichette
Errore di compilazione. Durante la compilazione, questo messaggio significa che sono state utilizzate più di 255 etichette di riga in un'istruzione ON...GOSUB o ON...GOTO.

Troppe variabili per INPUT
Errore di compilazione. Un'istruzione INPUT è limitata a 60 variabili.

Troppe variabili per LINE INPUT
Errore di compilazione. In un'istruzione LINE INPUT è consentita solo una variabile.

Troppi argomenti in una chiamata di funzione
Errore di compilazione. Le chiamate di funzione sono limitate a 60 argomenti.

Troppi file
Errore di esecuzione. Questo errore significa che il programma ha cercato di creare un nuovo file (con SAVE o OPEN) in una directory che conteneva già 255 file (67).

TYPE senza END TYPE
Errore di compilazione. A un'istruzione TYPE manca la relativa istruzione END TYPE.

Un identificatore non può finire per %, &, !, #, o $
Errore di compilazione. Questi suffissi non sono consentiti negli identificatori dei tipi, nei nomi dei sottoprogrammi o nei nomi che appaiono nelle istruzioni COMMON.

Un identificatore non può includere un punto
Errore di compilazione. I nomi dei tipi definiti dall'utente e quelli degli elementi dei record non possono contenere un punto. Un punto, quando viene utilizzato con le variabili record, può separare soltanto il nome del record e il nome dell'elemento. Un punto non può essere utilizzato se la parte del nome che lo precede corrisponde al nome di una variabile esistente definita con la clausola AS tipo utente.

Valori dell'istruzione DATA esauriti
Si è cercato di eseguire un'istruzione READ, ma le istruzioni DATA non contengono più alcuna voce da leggere (4).

Variabile di conteggio FOR già in uso
Errore di compilazione. Questo errore si verifica quando una variabile di conteggio viene utilizzata più di una volta in cicli nidificati FOR.

Variabile di conteggio FOR non valida
Errore di compilazione. Questo errore è di solito provocato dall'utilizzo di una variabile di tipo errato come contatore di un ciclo FOR (la variabile deve essere di tipo numerico semplice).

Variabili di tipo utente non ammesse nell'espressione
Errore di compilazione. Non è possibile utilizzare una variabile record nelle istruzioni in questione.

WEND senza WHILE
Errore di compilazione. A un'istruzione WEND non corrisponde un'istruzione WHILE.

WHILE senza WEND
Errore di compilazione. A un'istruzione WHILE non corrisponde un'istruzione WEND.

Indice analitico

! (tipo di dati a precisione semplice in virgola mobile), 6,16, 26, 97
" (stringa), 117
(creazione di musica), 336, 345
(tipo di dati a precisione doppia in virgola mobile), 6,16, 25, 99
$ (stringa a lunghezza variabile), 117
$ (tipo di dati stringa), 6, 16, 26, 97
$DYNAMIC, metacomando, 150, 157, 162-163
$STATIC, metacomando, 150, 157, 172-173
% (tipo di dati intero), 6, 16, 20, 97
& (tipo di dati intero lungo), 6, 16, 23, 97
' (commento), 486, 504
* (carattere jolly), 431
* (stringa a lunghezza fissa), 119
+ (creazione di musica), 336, 345
+ (movimento del cursore grafico), 293
+ (operatore di concatenazione), 56, 61, 64-65, 123
: (due punti), 486, 492-493
; (con istruzione INPUT), 215, 216
< (creazione di musica), 336, 344
< (operatore minore di), 56-57
<= (operatore minore o uguale a), 56-58
<> (operatore diverso da), 56-57
= (espressione stringa), 117
= (operatore uguale a), 56-58
> (creazione di musica), 336, 344
> (operatore maggiore di), 56-58
>= (operatore maggiore o uguale a), 56-57
? (carattere jolly), 431
\ (operatore di divisione intera), 61

ABS, funzione, 178, 182-184
Adattatore AT&T, 275-277
Adattatore Olivetti, 275-277
Adattatori, 275-277, 324-328
Adattatori video, 275-277, 324-328
Ambiente DOS, 488, 495-498
AND, operatore, 56, 59-60, 62-63
Animazione, 282, 308, 321-322
Apertura dei file, 409-410, 457-461
Apici nei commenti ('), 486, 504
APPEND, parola chiave, 410, 459-461
Arcotangente, 179, 184-185
Array, 149-157
accesso agli elementi, 152-153, 154-155
dichiarazione, 159-162
dimensioni, 151-153
dinamico, 93, 156-157, 164
elementi, 151-153
funzione LBOUND, 150, 165-166
funzione UBOUND, 150, 174-175
indici, 151-153
istruzione
DATA, 150, 154, 157-159
DIM, 150-152, 155, 159-162
ERASE, 150, 157, 163-165
OPTION BASE, 150, 152, 166-167
READ, 150, 154, 167-169
REDIM, 150, 157, 169-171
RESTORE, 150, 171-172

memorizzazione dei dati, 154
nei sottoprogrammi, 156
nelle funzioni, 156
statici e dinamici, 93, 156-157
stringa, 118-119, 128-129
tipi di dati definiti dall'utente, 155
variabili, 151-153
Arrotondamento, 181, 186-187
ASC, funzione, 117, 126-127
Asterisco (*), 119, 431
ATN, funzione, 178, 179, 184-85

BEEP, istruzione, 334, 340-341
BINARY, parola chiave, 410, 459
BLOAD, istruzione, 514, 515, 517-520
BSAVE, istruzione, 514, 515, 520-522
Buffer
carattere, 214, 217-219
di input, 393
di output, 393
file, 410, 458
memoria, 392-393
musica, 339-340, 342-344
video, 515, 521

CALL, istruzione, 84, 86, 88-90
CALL ABSOLUTE, istruzione, 513, 535-536, 538-540
Cancellazione della visualizzazione, 209-211
Caratteri di dichiarazione del tipo, 6-7, 16, 97
a precisione doppia in virgola mobile (#), 18
a precisione semplice in virgola mobile (!), 25
intero (%), 20
intero lungo (&), 23
stringa ($), 26
Caratteri jolly, 431
Casella motivo, 302-305
Cavo null modem, 395
CDBL, funzione, 178, 181, 185-186
CGA (Color Graphics Adapter), 275-277
CHAIN, istruzione, 84, 87, 90-92, 487, 507
CHDIR, istruzione, 486, 489-490
Chiusura dei file, 411, 418
CHR$, funzione, 117, 127-129
Cicli, 35-38
istruzione DO...LOOP, 34, 37-38, 39-41
istruzione FOR...NEXT, 34, 36, 43-46
istruzione WHILE...WEND, 34, 36-37, 52-54
test finale, 38, 39, 40
test iniziale, 37, 38, 39, 40
uscita, 38, 41-43
CINT, funzione, 178, 181, 186-187
CIRCLE, istruzione, 276, 279, 283-286
CLEAR, istruzione, 486, 487-488, 490-492
CLNG, funzione, 178, 181, 187-189
CLOSE, istruzione, 409, 411, 418-419
CLS, istruzione, 206, 209-211
Codici di errore, 565
COLOR, istruzione, 276, 286-292
Colori
casella motivo, 302-305
e istruzione COLOR, 286-292
modifica, 305-308
negli adattatori, 277
nell'istruzione DRAW, 294-295
COM OFF, istruzione, 392, 395, 396-398
COM ON, istruzione, 392, 395, 396-398
COM STOP, istruzione, 392, 394, 396-398
COM1: e COM2:, porte seriali, 391-392, 395
e funzione EOF, 425
e funzione LOC, 442
e funzione LOF, 446
Commenti, 486, 493, 503-504
COMMON, istruzione, 84, 87, 90, 92-96, 156, 507
Comunicazioni, 391-395
buffer, 392, 393
caricamento caratteri, 394
cavo null modem, 395
definizione dei collegamenti, 392-393
gestione degli eventi, 394-395, 398-400
invio/ricezione caratteri, 394
istruzione COM OFF, 392, 395, 396-398

istruzione COM ON, 392, 394, 396-398
istruzione COM STOP, 392, 394, 396-398
istruzione ON COM GOSUB, 392, 394, 398-400
istruzione OPEN COM, 392, 393, 400-405
memoria, 391, 392
parametri, 393, 400-405
Concatenamento, 61, 64-65, 87, 90-92, 123
CONST, istruzione, 4-5, 11-14, 158
Controllo del flusso, 33-38
cicli, 35-38
istruzione DO...LOOP, 34, 37-38, 39-41
istruzione FOR...NEXT, 34, 36, 43-46
istruzione GOTO, 34, 46-48
istruzione ON...GOSUB, 34, 35, 48-50
istruzione ON...GOTO, 34, 35, 50-52
istruzione WHILE...WEND, 34, 36-37, 52-54
istruzioni EXIT, 34, 38, 41-43
salto, 33-35, 46-52, 86, 104-106
Conversione numerica, 180-181
da stringhe, 125, 145-147
in stringhe, 125, 140-141
in valori a precisione doppia in virgola mobile, 180-181, 185-186
in valori a precisione semplice in virgola mobile, 181, 190-191
in valori interi, 181, 186-187
in valori interi lunghi, 181, 187-189
per l'istruzione FIELD, 412-413, 419-424, 450-454
tra binari ed esadecimali, 97-98
Coordinate dello schermo
e funzione PMAP, 281, 310-313
e istruzione VIEW, 280, 328-330
e istruzione WINDOW, 281, 330-332
grafici, 278-282
joystick, 354-355, 362-364
penna ottica, 354, 358-361
COS, funzione, 178, 179, 189-190
Coseno, 179, 189-190
Costanti, 4-5, 11-14
stringa, 118
valori ed espressioni, 5
CSRLIN, funzione, 206, 208, 211-212
Cursore
grafico, 293, 314
testo, 208, 211-212, 221-225
CVD, funzione, 409, 419-421
CVDMBF, funzione, 409, 412, 421-422
CVI, funzione, 409, 422
CVL, funzione, 409, 422-423
CVS, funzione, 409, 423-424
CVSMBF, funzione, 409, 412, 424-425

Data e ora, 259-261
funzione DATE$, 260, 262-264
funzione TIME$, 260, 267-269
funzione TIMER, 260, 269-270
impostazione dell'ora, 261, 267-269
impostazione della data, 259-260, 262-264
istruzione DATE$, 259, 260, 262-264
istruzione ON TIMER GOSUB, 260, 261, 264-265
istruzione SLEEP, 260, 261, 265-267
istruzione TIME$, 260, 267-269
istruzioni TIMER ON/OFF/STOP, 260, 261, 270-271
pausa nei programmi, 261, 265-267
visualizzazione data attuale, 259-260, 262-264
visualizzazione ora attuale, 260-261, 267-269
DATA, istruzione, 150, 157-159, 535, 537
DATE$, funzione, 260, 262-264
DATE$, istruzione, 259, 260, 262-264
Debug
comando Analizza il flusso, 559-560
comando Elimina ogni punto di interruzione, 555
comando Imposta istruzione successiva, 556-557
comando Passo, 557-558
comando Procedura passo, 555-556
comando Punto di interruzione, 558-559
TROFF, istruzione, 560-561

TRON, istruzione, 560-561
DECLARE, istruzione,
84, 85, 86, 88, 96-98
DEF FN, istruzione, 84, 87, 98-101
DEF SEG, istruzione, 515, 536, 540-542
DEFtipo, istruzione, 4, 7, 13, 14-16
DGROUP, 94, 119, 515
DIM, istruzione,
84, 150-152, 155, 159-162
Dimensioni dello schermo, 208, 236-238
Directory, 417, 489
concatenamento, 489-490
Directory creazione, 502-503
eliminazione, 504-506
spostamento dei file, 455-456
visualizzazione dei nomi dei file,
430-432
DO...LOOP, istruzione, 34, 37-38, 39-41
Doppi apici nelle stringhe ("), 117
DOS, 487, 508-509
DOUBLE, parola chiave, 4, 17-18
DRAW, istruzione, 276, 282, 292-296
Driver di dispositivo, 488-489, 498-500
Due punti (:), 486, 492-493

E commerciale (&), 6, 16, 23, 97
EGA (Enhanced Graphics Adapter),
275-277
END, istruzione, 486, 487, 494
ENVIRON, istruzione, 486, 488, 495-496
ENVIRON$, funzione, 486, 488, 497-498
EOF, funzione, 409, 416, 425-426
EQV, operatore, 56, 59, 65-67
ERASE, istruzione,
150, 157, 163-165, 491
ERDEV, funzione, 242, 243, 244-246
ERDEV$, funzione, 242, 243, 247-248
ERL, funzione, 242, 243, 248-249
ERR, funzione, 242, 243, 249-250, 564
ERROR, istruzione 242, 243, 250-252
Errori di compilazione, 242
Errori di sintassi, 242
Esecuzione dei programmi, 487, 506-507
Esponenti, 180, 191-192
Espressione stringa, 117
EXIT, istruzioni, 34, 38, 41-43
EXP, funzione, 178, 180, 191-192

Farheap, 515-516, 522
FIELD, istruzione, 409, 412-413, 426-428
File, 407-417
accesso a, 459
ad accesso casuale, 408, 410, 412-413,
433-435, 459-461, 466-467
lettura da, 413, 433-435
scrittura nei, 414, 466-467
apertura, 408-411, 457-461
binari, 408, 410, 413-414, 436-453, 459,
466-467
lettura da, 413-414, 436-453
scrittura nei, 414, 466-467
chiusura, 411, 418
CVD, funzione, 409, 419-421
CVDMBF, funzione, 409, 412, 421-422
CVI, funzione, 409, 422
CVL, funzione, 409, 422-423
CVS, funzione, 409, 423-424
CVSMBF, funzione, 409, 412, 424-425
dimensioni, 416, 446
funzioni
EOF, 409, 416, 425-426
FILEATTR, 409, 416, 428-430
FREEFILE, 408, 409, 432-433
INPUT$, 409, 414-415, 438-440
LOC, 409, 416, 442-444
LOF, 409, 416, 446
LTRIM$, 409, 415, 449-450
MKD$, 409, 450-451
MKDMBF$, 409, 412, 451-452
MKI$, 409, 452-453
MKL$, 410, 453
MKS$, 410, 453-454
MKSMBF$, 410, 412, 454-455
RTRIM$, 410, 415, 472
SEEK, 410, 416, 472-474
istruzioni
CLOSE, 409, 411, 418-419
FIELD, 409, 412-413, 426-428
FILES, 409, 417, 430-432
GET, 409, 413, 433-436
INPUT #, 409, 413, 414, 436-438
LINE INPUT #,
409, 413, 414, 440-442
LOCK, 409, 417, 444-445
LSET, 409, 412, 446-449

NAME, 410, 417, 455-456
OPEN, 410, 411, 457-461
PRINT #, 410, 415, 461-464
PRINT USING #, 410, 464-466
PUT, 410, 414, 466-468
RESET, 410, 468-469
RSET, 410, 412, 469-472
SEEK, 410, 416, 474-476
UNLOCK, 410, 417, 476-478
WIDTH, 410, 417, 478-480
WRITE #, 410, 415, 480-482
lettura dai, 413-415, 433-442
lunghezza dei, 416, 446
numeri di file, 408, 432-433
parole chiave
APPEND, 410, 459-461
BINARY, 410, 459
INPUT, 459
OUTPUT, 410, 459
RANDOM, 410, 459
puntatore, 416, 425, 442
reti, 411, 417, 444-445, 476-478
rinomina, 417, 455-456
scrittura nei,
414, 415, 461-468, 480-482
sequenziali, 408, 410, 413-414, 417,
436-442, 459-464, 478-482
impostazione larghezza riga,
417, 478-480
lettura da, 413-414, 436-442
scrittura nei, 415, 461-464, 480-482
FILEATTR, funzione, 409, 416, 428-430
FILES, istruzione, 409, 417, 430-432
FOR...NEXT, istruzione, 34, 36, 43-46
FRE, funzione, 514, 515, 522-524
FREEFILE, funzione, 410, 432-433
FUNCTION, istruzione,
84, 86, 87, 97, 101-104
Funzione FIX, 178, 181, 192-193
Funzioni
array, 156
definite dall'utente, 86-87, 98-104
trigonometriche, 177-179
arcotangente, 179, 184-185
coseno, 179, 189-190
seno, 179, 200-201
tangente, 179, 202-203
Gestione degli eventi, 244
joystick, 352-354, 357-358
musica, 339, 342
nelle comunicazioni, 394-395, 398-400
penna ottica, 352-354, 356-357
tastiera, 371, 372, 378-380
timer, 261, 264-265
Gestione dei programmi, 486-489
(:) due punti, 486, 492-493
commenti, 486, 493, 503-504
driver di dispositivo, 488-489, 498-500
esecuzione comandi del DOS,
487, 508-509
esecuzione dei programmi,
487, 506-507
funzione ENVIRON$,
486, 488, 497-498
funzione IOCTL$, 486, 489, 499-500
istruzioni
CHDIR, 486, 489-490
CLEAR, 486, 487-488, 490-492
END, 486, 487, 494
ENVIRON, 486, 488, 495-496
IOCTL, 486, 488, 498-499
KILL, 486, 500-502
MKDIR, 486, 489, 502-503
REM, 486, 503-504
RMDIR, 486, 489, 504-506
RUN 486, 487, 506-507
SHELL, 486, 487, 508-509
STOP, 486, 510
SYSTEM, 486, 511
uscita dai programmi,
487, 494, 510, 511
variabili di ambiente, 488, 495-498
Gestori degli errori, 243
funzione ERDEV, 242, 243, 244-246
funzione ERDEV$, 242, 243, 247-248
funzione ERL, 242, 243, 248-249
funzione ERR, 242, 243, 249-250, 564
Gestori dei file, 416-417, 428-430
GET, istruzione
grafica, 276, 282, 296-298
I/O su file, 409, 413, 433-436
GOSUB...RETURN, istruzione
84, 86, 104-106
GOTO, istruzione 34, 46-48

Grafici, 275-283
  attributi di visualizzazione, 280-286
  casella motivo, 301-304
  colori, 277, 280, 286-292, 294-295, 305-308
  comando colore nell'istruzione DRAW, 294-295
  coordinate dello schermo, 278, 310-313, 328, 330-332
  cursore, 293, 314
  fattore di scala, 293
  funzione PMAP, 276, 281, 310-313
  funzione POINT, 276, 279, 313-315
  funzione SCREEN, 276, 322-323
  istruzioni
    CIRCLE, 276, 279, 283-286
    COLOR, 276, 286-292
    DRAW, 276, 282, 292-296
    GET, 276, 282, 296-298
    LINE, 276, 280, 298-301
    PAINT, 276, 279-280, 301-305
    PALETTE, 276, 305-308
    PALETTE USING, 276, 305-308
    PCOPY, 276, 282, 308-310
    PRESET, 276, 279, 315-317
    PUT, 276, 282, 317-322
    SCREEN, 276, 278, 323-328
    VIEW, 276, 280, 328-330
    WINDOW, 276, 281, 310-313, 330-332
  modalità video, 277-278, 286-292, 324-328
  pagine video, 282, 308-310, 324-325
  pixel, 277, 313-317
  rapporto, 284-285
  risoluzione, 277
  rotazione, 293-294
  viewport, 280, 328-330

Heap, 515-516, 522
HEX$, funzione, 4, 7, 18-19
HGC (Hercules Graphics Card), 277

IF, istruzione, 56, 67-70
IMP, operatore, 56, 59, 71-72
Inizializzazione delle variabili, 7, 21-22
INKEY$, funzione, 206, 212-214
INP, funzione, 514, 516, 524-526
INPUT, istruzione, 206, 207, 215-217
INPUT, parola chiave, 410, 459
INPUT #, istruzione, 409, 413, 414, 436-438
INPUT$, funzione
  I/O su file, 409, 414-415, 438-440
  tastiera, 206, 207, 217-219
Input/output 206-209
  cancellazione visualizzazione, 209-211
  dimensioni dello schermo, 208, 236-238
  funzioni
    CSRLIN, 206, 208, 211-212
    INKEY$, 206, 212-214
    INPUT$, 206, 207, 217-219
    POS, 206, 208, 223-225
    SPC, 206, 207, 231-232
    TAB, 206, 207, 232-234
  input da tastiera, 206, 212-221
  istruzioni
    CLS, 206, 209-211
    INPUT, 206, 215-217
    LINE INPUT, 206, 220-221
    LOCATE, 206, 208, 221-223
    PRINT USING, 206, 207, 227-231
    PRINT, 207, 225-227
    VIEW PRINT, 207, 209, 234-236
    WIDTH, 206, 208, 236-238
    WRITE, 206, 207, 238-239
  output di testo, 207, 225-234, 238-239
  porte, 516
  posizione del cursore, 207, 211-212
  viewport testo, 209, 234-236

INSTR, funzione, 117, 123, 129-131
INT, funzione, 178, 181, 193-194
INTEGER, parola chiave, 4, 19-20
Intercettazione degli errori, 242-243
  funzione ERDEV, 242, 243, 244-246
  funzione ERDEV$, 242, 243, 247-248
  funzione ERL, 242, 243, 248-249
  gestori degli errori, 243, 244-250
  istruzione ERROR, 242, 243, 250-252, 564
  istruzione ON ERROR GOTO, 242, 243, 253-255, 564

istruzione RESUME, 242, 243, 255-257
IOCTL, istruzione, 486, 488, 498-499
IOCTL$, funzione, 486, 489, 499-500

Joystick, 352
coordinate, 354-355, 362-364
funzione STICK, 352, 354, 362-364
funzione STRIG, 352, 355, 365-366
gestione degli eventi, 352-354, 357-358
istruzione ON STRIG GOSUB, 352, 357-358
istruzioni STRIG ON/OFF/STOP, 352, 354, 366-368

KEY istruzione, 371, 373-374
KEY LIST istruzione, 371-372
KEY OFF, istruzione, 373
KEY ON istruzione, 372, 373
KEY(n)ON/OFF/STOP istruzioni, 371, 372, 374-378
KILL, istruzione, 486, 500-502
LBOUND, funzione, 150, 165-166
LCASE$, funzione, 117, 124, 131-132
LEFT$, funzione, 117, 122, 132-133
LEN, funzione, 117, 121, 133-135
LET, istruzione, 4, 21-22
LINE, istruzione, 280, 298-301
LINE INPUT, istruzione, 206, 220-221
LINE INPUT #, istruzione, 409, 413, 414, 440-442
Linguaggio macchina, 551-554
conversione in istruzioni DATA, 535, 537
funzione VARPTR, 536, 542-544
funzione VARSEG, 536, 544-545
funzione VARTPR$, 536, 542-544
istruzione CALL ABSOLUTE, 535-536, 538-540
istruzione DEF SEG, 536,540-542
LOC, funzione, 409, 416, 442-444
LOCATE, istruzione, 206, 208, 221-223
LOCK, istruzione, 407, 417, 444-445
LOF, funzione, 409, 416, 446
LOG, funzione, 178, 180, 194-196
Logaritmi, 180, 191-192, 194-196
Logaritmi naturali, 180, 194-196
LONG, parola chiave, 4, 22-23
LPOS, funzione, 382, 383-386
LPRINT USING, istruzione, 381-382, 388-389
LPRINT, istruzione, 381-383, 386-388
LSET, istruzione, 409, 412, 446-449
LTRIM$, funzione, 409, 415, 449-450

Matematica
arrotondamento, 181, 186-187
esponenti, 180, 191-192
funzioni
ABS, 178, 182-184
ATN, 178, 179, 184-185
CDBL, 178, 181, 185-186
CINT, 178, 181, 186-187
CLNG, 178, 181, 187-189
COS, 178, 179, 189-190
CSNG, 190-191
CSRLIN, 206, 208, 211-212
di conversione numerica, 180-181, 185-189, 190-191
EXP, 178, 180, 191-192
FIX, 178, 181, 192-193
INT, 178, 181, 193-194
LOG, 178, 180, 194-196
RND, 178, 181, 197-199
SGN, 178, 199-200
SIN, 178, 179, 200-201
SQR, 178, 179, 201-202
TAN, 178, 179, 202-203
trigonometriche, 177-180, 184-185, 189-190, 200-201, 202-203
istruzione RANDOMIZE, 178, 181, 196-197
logaritmi, 180, 191-192, 194-196
logaritmi naturali, 180, 194-196
numeri casuali, 181-182, 196-199
radice quadrata, 179, 201-202
segni, 199-200
troncamento, 181, 192-194
valori assoluti, 182-184
MCGA (Multi-Color Graphics Array), 275-277
MDPA (Monochrome Display and Printer Adapter), 277
Memoria, 513-517
accesso alla, 514-515, 528-531

- buffer, 392, 393
- caricamento dei blocchi di, 515, 517-520
- e routine di linguaggio macchina, 535-536, 542-544
- funzione FRE, 514, 515, 522-524
- funzione INP, 514, 516, 524-526
- funzione PEEK, 514, 515-516, 528-529
- heap, 515-516, 522
- indirizzamento, 514
- istruzione BLOAD, 514, 515, 517-520
- istruzione BSAVE, 514, 515, 520-522
- istruzione OUT, 514, 516, 526-527
- istruzione POKE, 514, 515-516, 530-531
- istruzione WAIT, 514, 516, 531-533
- salvataggio dei blocchi, 515, 520-522
- stack, 487, 491, 516, 522

Metacomandi
- $DYNAMIC, 150, 157, 162-163
- $STATIC, 150, 157, 172-173
- introduzione, 503-504

MID$, funzione, 117, 123, 135-136
MID$, istruzione, 117, 124, 136-137
MKD$, funzione, 409, 450-451
MKDIR, istruzione, 436, 489, 502-503
MKDMBF$, funzione, 409, 412, 451-452
MKI$, funzione, 409, 452-453
MKL$, funzione, 410, 453
MKS$, funzione, 410, 453-454
MKSMBF$, funzione, 410, 412, 454-455
MOD, operatore, 56, 61, 72-73
Modalità video, 277-278, 286-292, 324-328
Mouse, penne ottiche come, 355
Musica, 333-340
- durata, 336-338, 345
- funzione PLAY, 334, 342-344
- in primo piano e sottofondo, 339-340, 345-346
- istruzione ON PLAY GOSUB, 334, 339, 341-342
- istruzione PLAY, 334-339, 344-347
- istruzioni PLAY ON/OFF/STOP, 334, 339, 340, 347-349

NAME, istruzione, 410, 417, 455-456
NOT, operatore, 56, 59, 73-75
Numeri
- casuali, 181-182, 196-199
- di file, 408, 432-433
- esadecimali, 7-8, 18-19
- ottali, 7-8, 24-25
- PRINT USING, caratteri di formattazione, 230
- segno dei, 199-200
- troncamento, 181, 192-194
- valore assoluto, 182-184

OCT$, funzione, 4, 8, 24-25
ON COM GOSUB, istruzione, 392, 394, 398-400
ON ERROR GOTO, istruzione, 34, 242, 243, 253-255
ON KEY(n) GOSUB, istruzione, 372, 378-380
ON PEN GOSUB, istruzione, 352, 353, 356-357
ON PLAY GOSUB, istruzione, 334, 339, 341-342
ON STRIG GOSUB, istruzione, 352, 357-358
ON TIMER GOSUB, istruzione, 260, 261, 264-265
ON...GOSUB, istruzione, 34, 35, 48-50
ON...GOTO, istruzione, 34, 35, 50-52
OPEN, istruzione, 410, 411, 457-461
OPEN COM, istruzione, 392, 393, 400-405
Operatore
- di concatenamento (+), 56, 61, 64-65
- di divisione intera (\), 61
- diverso da (<>), 56-57
- maggiore di (>), 56-58
- maggiore o uguale a (>=), 56-58
- minore di (<), 56-57
- minore o uguale a (<=), 56-58
- uguale a (=), 56-58

Operatori, 56-61
- AND, 56, 59, 62-63
- con istruzione IF, 68

concatenamento (+), 56, 61, 64-65
decisionali
istruzione IF, 56, 67-70
istruzione SELECT CASE, 56, 60, 77-80
AND, 56, 59, 62-63
di concatenamento (+), 56, 61, 64-65
EQV, 56, 59, 65-67
IMP, 56, 59, 71-72
MOD, 56, 61, 72-73
NOT, 56, 59, 73-75
OR, 56, 59-60, 75-76
XOR, 56, 59, 80-82
logici, 59-60
relazionali, 56-59
con le istruzioni IF, 68
diverso da (<>), 56-57
EQV, 56, 59, 65-67
maggiore di (>), 56-58
maggiore o uguale a (>=), 56-57
minore di (<), 56-57
minore o uguale a (<=), 56-58
uguale a (=), 56-58
OPTION BASE, istruzione, 150, 152, 166-167
OR, operatore, 56, 59-60, 75-76
OUT, istruzione, 514, 516, 526-527
Output di testo, 207, 225-234, 238-239
OUTPUT, parola chiave, 410, 459

Pagine video, 282, 308-310, 324-325
PAINT, istruzione, 276, 279-280, 301-305
PALETTE, istruzione, 276, 305-308
PALETTE USING, istruzione, 276, 305-308
Parentesi angolari (<>), 56-57
PCOPY, istruzione, 276, 282, 308-310
PEEK, funzione 514, 515-516, 528-529
PEN, funzione, 352, 354, 358-361
PEN ON/OFF/STOP, istruzioni, 352, 353, 361-362
Penne ottiche, 351, 360
coordinate, 354-355, 358-361
funzione PEN, 352, 354, 358-361
gestione degli eventi, 352-354, 356-357
istruzione ON PEN GOSUB, 352, 353, 356-357
istruzioni PEN ON/OFF/STOP, 352, 353, 361-362
mouse come, 355
Pixel, 277
bit per pixel, 298
disegno, 278-280, 315-317
rapporto, 284-285
restituzione delle informazioni da, 279, 313-315
PLAY, funzione, 334, 342-344
PLAY, istruzione, 334, 334-339, 344-347
PLAY ON/OFF/STOP, istruzioni, 334, 339, 340, 347-349
PMAP, funzione, 276, 281, 310-313
POINT, funzione, 276, 279, 313-315
POKE, funzione, 514, 515-516, 528-529
Porte, 513, 516
funzione INP, 514, 516, 524-526
funzione POS, 206, 208, 223-225
istruzione OUT, 514, 516, 526-527
istruzione WAIT, 514, 516, 531-533
seriali, 392-393
e funzione EOF, 425
e funzione LOC, 442
e funzione LOF, 446
PRESET, istruzione, 276, 279, 315-317
PRINT, istruzione, 207, 225-227
PRINT #, istruzione, 410, 415, 461, 464
PRINT USING, istruzione, 206, 207, 227-231
PRINT USING #, istruzione, 410, 464-466
Procedure, 83-87
concatenamento, 87, 90-92
funzioni definite dall'utente, 86-87, 98-104
istruzione CALL, 84, 86, 88-90
istruzione CHAIN, 84, 87, 90-92
istruzione COMMON, 84, 87, 90, 92-96
istruzione DECLARE, 84, 85, 86, 88, 96-98
istruzione DEF FN, 84, 87, 98-101
istruzione FUNCTION, 84, 86, 87, 97, 101-104
istruzione GOSUB...RETURN, 84, 86, 104-106
istruzione SHARED, 84, 87, 103, 107-108

istruzione STATIC,
84, 85, 87, 103, 109-111
istruzione SUB, 84, 85, 111-114
ricorsive, 112
salto, 86, 104-106
sottoprogrammi, 85-86, 111-114
subroutine, 86, 104-106
Programmi di uscita, 487, 494, 510-511
Programmi pausa dei, 261, 265-267
PSET, istruzione, 276, 279, 315-317
Puntatore del file, 416, 425, 442
Punto di domanda (caratteri jolly), 421
Punto e virgola (;), 215, 216
Punto esclamativo (!), 6, 16, 26, 97
PUT, istruzione
grafica, 276, 282, 317-322
I/O su file, 410, 414, 466-468

Radice quadrata, 179, 201-202
RANDOM, parola chiave, 410, 459
RANDOMIZE, istruzione,
178, 181, 196-197
Rapporto, 284-285
READ, istruzione, 150, 154, 167-169
REDIM, istruzione, 150, 157, 169-171
REM, istruzione, 486, 503-504
RESET, istruzione, 410, 468-469
RESTORE, istruzione, 150, 171-172
RESUME, istruzione, 242, 243, 255-257
Reti, 411, 417, 444-445, 476-478
Riavvio dei programmi, 487, 506-507
RIGHT$, funzione, 117, 122, 138-139
RMDIR, istruzione, 486, 489, 504-506
RND, funzione, 178, 181, 197-199
Rotazione dei grafici, 293-294
RSET, funzione 410, 412, 469-472
RTRIM$, funzione, 410, 415, 472
RUN, istruzione, 487, 506-507

Salto
condizionato, 34-35, 48-52
incondizionato, 34, 46-48, 86, 104-106
istruzione GOSUB...RETURN,
86, 104-106
istruzione GOTO, 46-48
istruzione ON...GOSUB, 48-50
istruzione ON...GOTO, 50-52
SCREEN, istruzione, 276, 278, 323-328
SCREEN, funzione, 276, 322-323
SEEK, funzione, 410, 416, 472-474
SEEK, istruzione, 410, 416, 474-476
Segno di dollaro ($), 6, 16, 26, 97, 117
Segno di percentuale (%), 6, 16, 20, 97
Segno di uguale (=), 117
SELECT CASE, istruzione,
35, 56, 60, 77-80
Seno, 179, 200-201
SGN, funzione, 178, 199-200
SHARED, istruzione,
84, 87, 103, 107-108
SHARED, parola chiave, 84, 92, 94
SHELL, istruzione, 486, 487, 508-509
SIN, funzione, 178, 179, 200-201
SINGLE, parola chiave, 4, 25-26
SLEEP, istruzione, 260, 261, 265-267
Sottoprogrammi, 85, 86, 111-114
array, 156
dichiarazione, 84, 96-98, 112
richiamo, 85, 88-90, 112
ricorsivi, 112
uscita, 85
SPACE$, funzione, 117, 123, 139-140
SPC, funzione, 206, 207, 231-232
SQR, funzione, 178, 179, 201-202
Stack, 487, 491, 516, 522
Stampante, 381-383
funzione LPOS, 382, 383, 383-386
istruzione LPRINT USING,
381-382, 388-389
Stampante istruzione LPRINT,
381-383, 386-388
istruzione WIDTH LPRINT,
381-383, 390
STATIC, istruzione,
84, 85, 87, 103, 109-111, 157
STICK, funzione 352, 354, 362-364
STOP, istruzione, 486, 510
STR$, funzione, 117, 125, 140-141
STRIG, funzione, 352, 355, 365-366
STRIG ON/OFF/STOP, istruzioni,
352, 354, 366-368
STRING, parola chiave, 4, 26-27

STRING$, funzione, 117, 124, 141-143
Stringhe, 116-125
  a lunghezza fissa, 119-120, 220-221
  a lunghezza variabile, 119
  concatenamento, 61, 64-65, 123
  confronto, 121
  conversione dei numeri da,
    125, 146-147
  conversione dei numeri in, 125, 140-141
  definizione, 116-120
  estrazione di parti di, 122-123, 133-135
  funzioni
    ASC, 117, 126-127
    CHR$, 117, 127-129
    INSTR, 117, 123, 129-131
    LCASE$, 117, 124, 131-132
    LEFT$, 117, 122, 132-133
    LEN, 117, 121, 133-135
    MID$, 117, 123, 135-136
    RIGHT$, 117, 122, 138-139
    SPACE$, 117, 123, 139-140
    STR$, 117, 125, 140-141
    STRING$, 117, 124, 141-143
    UCASE$, 117, 124, 144-145
    VAL, 117, 125, 145-147
  individuazione caratteri di,
    123, 129-131, 135-136, 138-139
  istruzione MID$, 117, 124, 136-137
  memorizzazione, 119
  modifica di parti di, 124, 136-137
SUB, istruzione, 84, 85, 111-114
Subroutine, 86, 104-106
Suono, 333-340
  durata, 336-338, 345
  funzione PLAY, 334, 342-344
  in primo piano e sottofondo,
    339-340, 345-346
  istruzione BEEP, 334, 340-341
  istruzione ON PLAY GOSUB,
    334, 339, 341-342
  istruzione PLAY,
    334, 334-339, 344-347
  istruzione SOUND, 334, 340, 349-350
  istruzioni PLAY ON/OFF/STOP,
    334, 339, 340, 347-349
SWAP, istruzione, 4, 8, 27-28
SYSTEM, istruzione, 486, 511

TAB, funzione, 206, 207, 232-234
TAN, funzione, 178, 179, 202-203
Tangente, 179, 202-203
Tasti rapidi, 371-372, 373-374
Tastiere, 371-372
  caricamento dell'input da, 206, 212-221
  convalidazione dell'input da, 94-96
  gestione degli eventi, 371, 372, 378-380
  istruzione KEY, 371, 373-374
  istruzione KEY LIST, 372-373
  istruzione KEY OFF, 373
  istruzione KEY ON, 372-373
  istruzione ON KEY(n) GOSUB;
    372, 378-380
  istruzioni KEY(n)OF/ON/STOP,
    372, 373, 374-378
  tasti rapidi, 371-372, 373-374
Termine dei programmi,
  487, 494, 510-511
TIME$, funzione, 260, 267-269
TIME$, istruzione, 260, 267-269
TIMER, funzione, 260, 269-270
TIMER ON/OFF/STOP, istruzioni,
  260, 261, 270-271
Tipi di dati definiti dall'utente, 8, 11, 29-31
  negli array, 155
TROFF; istruzione, 560-561
TRON, istruzione, 560-561
TYPE...END TYPE, istruzione, 4, 8-11,
  16, 29-31, 84, 155, 412, 413, 427

UBOUND, funzione, 150, 174-175
UCASE$, funzione, 117, 124, 144-145
UNLOCK, istruzione, 410, 417, 476-478
VAL, funzione, 117, 125, 145-147

Variabili, 3-11
  complesse, 8-11, 29-31
  comuni, 84, 87, 90, 92-96
  di ambiente, 488, 495-498
  dichiarazione, 5-7
  funzione HEX$, 4, 7, 18-19
  funzione OCT$, 4, 8, 24-25
  inizializzazione, 7, 21-22

istruzione CONST, 4-5, 11-14, 158
istruzione DEFtipo 4, 7, 13, 14-15
istruzione LET, 4, 21-22
istruzione SWAP, 4, 8, 27-28
istruzione TYPE...END TYPE, 4, 8-11, 16, 29-31
negli array, 151-153
parola chiave DOUBLE 4, 17-18
parola chiave INTEGER, 4, 19-20
parola chiave LONG, 4, 22-23
parola chiave SINGLE, 4, 25-26
parola chiave STRING, 4, 26-27
record, 8, 30, 448-449, 450-451
statiche, 85, 87, 103, 109-111
stringa, 5, 7, 116-118
e istruzione LSET, 447-448
e istruzione PLAY, 339-346
e istruzione RSET, 471
tipi, 3-7
VARPTR, funzione, 536, 542-544
VARPTR$, funzione, 536, 542-544
VARSEG, funzione, 536, 544-545
VGA (Video Graphics Array), 275-277
VIEW, istruzione, 276, 280, 328-330
VIEW PRINT, istruzione, 207, 209, 234-236
Viewport
eliminazione, 209-211
grafici, 280, 328-330
testo, 209, 234-236
Virgola (,), 215, 216

WAIT, istruzione, 514, 516, 531-533
WHILE...WEND, istruzione, 34, 36-37, 52-54
WIDTH, istruzione
I/O su file, 410, 417, 478-480
schermo, 206-208, 236-238
WIDTH LPRINT, istruzione, 382-383, 390
WINDOW, istruzione, 276, 330-332
WRITE, istruzione, 206, 207, 238-239
WRITE #, istruzione, 410, 415, 480-482

XOR, operatore, 56, 59, 80-82

La McGraw-Hill pubblica in tutto il mondo centinaia di libri di informatica per lo studio, la professione e il tempo libero. La produzione in lingua italiana comprende:

MICROCALCOLATORI

88 386 0049-X	Murray W.H., Pappas C.H., *L'Assembler per l'80286/80386*
88 386 0009-0	Peckham H., Ellis Jr W., Lodi E., *Il BASIC e il Commodore 64 in pratica*
88 386 0008-2	Peckham H., *Il BASIC e il PC-IBM in pratica*
88 386 0030-9	Ettlin W., Solberg G., *Il BASIC Microsoft*
88 386 0150-X	Jones E., *Come usare dBASE IV*
88 386 0040-6	Barnes L., *Come usare il dBASE III*
88 386 0056-2	Jones E., *Come usare il dBASE III PLUS*
88 386 0026-0	Ettlin W., *Come usare il Multiplan*
88 386 0019-8	Baras E.M., *Come usare il Symphony*
88 386 0064-3	Ettlin W., *Come usare il WordStar 4.0*
88 386 0095-3	Matthews M.S., Matthews C., *Come usare Lotus 1-2-3 Release 3 Versione inglese*
88 386 0054-6	Brownstein M., *Come usare Microsoft Word*
88 386 0149-6	Perry G.M., *Come usare Microsoft Word 5*
88 386 0080-5	Cowart R., *Come usare Microsoft Works*
88 386 0072-4	Cobb S., *Come usare Reflex*
88 386 0148-8	Matthews M.S., Matthews B.C., *Come usare Wordstar 5.5*
88 386 0050-3	Andersen D., Cooper C., Dempsey B., *dBASE III in pratica*
88 386 0093-7	Carrabis J.D., *dBASE III PLUS Applicazioni in rete locale*
88 386 0039-2	Hixson A.C., *Excel Guida avanzata*
88 386 0046-5	Jones E., *Excel per personal MS-DOS*
88 386 0152-6	Comi A., *Elaborazioni grafiche con il personal computer*
88 386 0055-4	Heilborn J., *Guida al Commodore 128*
88 386 0001-5	Heilborn J., Talbott R., *Guida al Commodore 64*
88 386 0043-0	Byers T.J., *Guida al PC AT IBM*
88 386 0021-X	Graham L.J., Field T., *Guida al PC-IBM*
88 386 0076-7	Wood S., *Guida al Turbo Pascal 4.0*
88 386 0044-9	Kater D.A., Kater R.L., *Guida alle stampanti Epson*
88 386 0138-0	Etlin W., Solberg G., *Il GW-BASIC per il PC1 Olivetti Prodest*
88 386 0041-4	Ettlin W., Solberg G., *Il GW-BASIC per Personal Computer Olivetti*
88 386 0126-7	Biow L., *Introduzione a Quattro*
88 386 0085-6	Baeseler F., Heck B., *Introduzione al Desktop Publishing*
88 386 0037-6	Kruglinski D., *Introduzione al Framework*
88 386 0042-2	Watt D., *Il Logo per il Commodore 64*
88 386 0053-8	Kyd C.W., *Lotus 1-2-3: applicazioni finanziarie*
88 386 0164-X	Williams A.T., *Lotus 1-2-3 Release 2.2 Versione inglese*

88 386 0025-2	Morgan C., Waite M., *Il manuale 8086/8088*
88 386 0058-9	Cramer W., Kane G., *Il manuale MC 68000*
88 386 0137-2	Siechert C., Wood C., *Il manuale MS-DOS 3.20 per il PC1 Olivetti Prodest*
88 386 0034-1	Hoffman P., *Il manuale MSX*
88 386 0047-3	Siechert C., Wood C., *Il manuale MS-DOS 3.20*
88 386 0159-3	Siechert C., Wood C., *Il manuale MS-DOS 3.30*
88 386 0158-5	Siechert C., Wood C., *Il manuale MS-DOS 4*
88 386 0052-X	Hoffman P., Nicoloff T., *Il manuale MS-DOS per Personal Computer Olivetti*
88 386 0018-X	Hoffman P., Nicoloff T., *Il manuale MS-DOS*
88 386 0184-4	Weber D., *Il manuale Novell Netware*
88 386 0087-2	Jamsa K., *Il manuale OS/2*
88 386 0171-2	Sheldon T., *Il manuale Windows 3 Versione italiana*
88 386 0141-0	Hedtke J.V., *Microsoft Word 4 Guida avanzata*
88 386 0068-6	Matthews M.S., Matthews B.C., *PageMaker 3.0 per personal MS-DOS*
88 386 0146-1	Guerriero M., Zampariolo H., *Programmare in Fred con Framework II e III*
88 386 0033-3	Sand P.A., *Programmazione avanzata in Pascal*
88 386 0051-1	Carroll D.W., *Programmazione in Turbo Pascal*
88 386 0912-8	Carroll D.W., *Programmazione in Turbo Pascal (libro+disco)*
88 386 0060-0	Robinson P.R., *Programmazione in Turbo Prolog*
88 386 0048-1	Peckham H., Ellis Jr W., Lodi E., *Programmazione strutturata in BASIC*
88 386 0070-8	Jamsa K., Nameroff S., *Strumenti di programmazione in Turbo Pascal*
88 386 0919-5	Jamsa K., Nameroff S., *Strumenti di programmazione in Turbo Pascal (libro+disco)*
88 386 0077-5	Mosher F.E., Schneider D.I., *Turbo BASIC Elementi di programmazione*
88 386 0071-6	Schildt H., *Turbo C elementi di programmazione*
88 386 0073-2	Schildt H., *Turbo C Programmazione avanzata*
88 386 0143-7	Schildt H., *Turbo Pascal 4.0 Programmazione avanzata*
88 386 0075-9	Schildt H., *Turbo Pascal Programmazione avanzata*
88 386 0069-4	Schildt H., *Turbo Prolog 1.1 Programmazione avanzata*

GUIDE COMPLETE

88 386 0156-9	Johnson N., *Guida completa AutoCAD versione 10 inglese*
88 386 0082-1	Schildt H., *Guida completa C*
88 386 0066-X	Carrabis J.D., *Guida completa dBASE III PLUS*
88 386 0081-3	Jamsa K., *Guida completa DOS*

88 386 0147-X	Jamsa K., *Guida completa DOS Versione 3.30 italiana*
88 386 0065-1	Campbel M., *Guida completa Lotus 1-2-3*
88 386 0161-5	Campbell M., *Guida completa Lotus 1-2-3 Versione 2.2 inglese*
88 386 0099-6	Alderman E., *Guida completa Microsoft Word*
88 386 0130-5	Keogh J., *Guida completa Paradox 2.0*
88 386 0110-0	McCoy Y., *Guida completa Quattro*
88 386 0124-0	Schildt H., *Guida completa Turbo C*
88 386 0078-3	O'Brien S., *Guida completa Turbo Pascal*
88 386 0162-3	O'Brien S., *Guida completa Turbo Pascal Versione 5.5 inglese*
88 386 0157-7	Holt M., Birmele R., *Guida completa Ventura 2*
88 386 0123-2	Todd G., *Guida completa Videoscrittura 4*

INFORMATICA PROFESSIONALE

88 386 0160-7	Price Waterhouse, *Computer Virus Analisi, prevenzione, metodi di difesa*
88 386 0165-8	K.W. Christopher, *MS-DOS Manuale di programmazione*
88 386 0153-4	D.M. Kalman, *Il manuale dei linguaggi dBASE*
88 386 0154-2	R. Spence, *Programmare in Clipper*
88 386 0059-7	M. Liskin, *Programmmazione avanzata in dBASE III Plus*
88 386 0094-5	E. Iacobucci, *OS/2 Manuale di programmmazione*

GUIDE TASCABILI

88 386 0122-4	C. Gilbert, *Guida tascabile Wordstar Versione 5.5*
88 386 0103-8	K. Jamsa, *Guida tascabile OS\2*
88 386 0100-3	H. Schildt, *Guida tascabile C*
88 386 0105-4	H. Schildt, *Guida tascabile Turbo C*
88 386 0104-6	K. Jamsa, *Guida tascabile Turbo Pascal 4.0*
88 386 0089-9	M. Liskin, *Guida tascabile dBASE III Plus*
88 386 0098-8	M. Campbell, *Guida tascabile Lotus 1-2-3*
88 386 0102-X	M. K. Jamsa, *Guida tascabile DOS Versione 3.30*
88 386 0107-0	E. Jones, *Guida tascabile Paradox*
88 386 0106-2	S. Cobb, *Guida tascabile Quattro*
88 386 0108-9	E. Alderman, *Guida tascabile Microsoft Word*
88 386 0127-5	K. Jamsa, *Guida tascabile OS/2 API*

Ghidini Daniel

06-10-92

Questo volume, sprovvisto del talloncino a fronte, è da considerarsi copia saggio-campione gratuito fuori commercio. Fuori campo applicazione IVA ed esente da bolla di accompagnamento (art. 22 L. 67/1987, art. 2, lett. i D.P.R. 633/1972 e art. 4 n. 6 D.P.R. 627/1978).

Arnson Gemmell Henderson
MS-DOS QBasic
McGraw-Hill Libri Italia
88 386 0257-3

MS-DOS QBasic™

GUIDA DEL PROGRAMMATORE

MS-DOS QBasic è il BASIC fornito da Microsoft con la nuova release 5 del sitema operativo MS-DOS. Destinato a sostituire il "vecchio" GW-Basic esso offre una serie di caratteristiche di rilievo quali l'interfaccia utente rinnovata, online help, nuovi strumenti di editing e debugging.
Il libro è organizzato in modo da garantire un accesso immediato e una completa informazione su tutti i comandi e tutte le funzioni del linguaggio. Centinaia di listati, esempi e suggerimenti corredano il testo, facendone uno strumento indispensabile sia per il programmatore alle prime armi, sia per il professionista.

ISBN 88-386-0257-3
9 788838 602573

Lire 65.000 (i. i.)